MC/DC: Enabling easy-to-use safety-critical code coverage analysis with LLVM

Alan Phipps, Texas Instruments

2022 LLVM Developers' Meeting

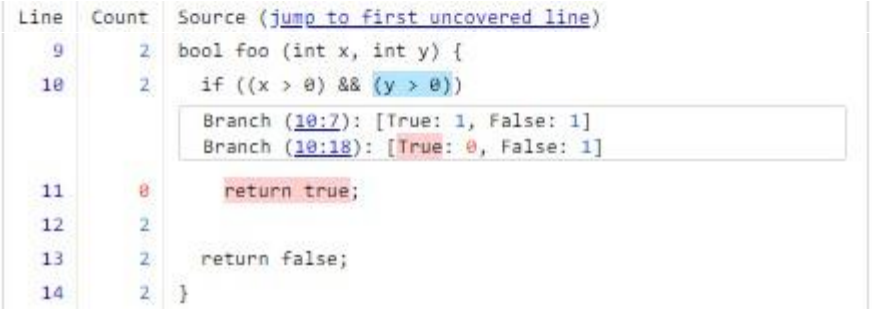
What is Source-based Code Coverage?

- A measurement for how thoroughly code has been executed during testing
  - Ideally all sections of code have an associated test
  - Un-executed code may be at higher risk of having lurking bugs

- Presently Supported Coverage criteria (in increasing level of granularity)
  - **Function**
    - Percentage of code functions executed at least once
  - **Line**
    - Percentage of code lines executed at least once
  - **Region**
    - Percentage of code statements executed at least once
  - **Branch (Recently added)**
    - Percentage of condition branches taken at least once

LLVM Coverage Visualization

- LLVM Coverage Utility (llvm-cov)



Line	Count	Source (jump to first uncovered line)
9	2	bool foo (int x, int y) {
10	2	if ((x > 0) && (y > 0))
		Branch (10:7): [True: 1, False: 1]
		Branch (10:18): [True: 0, False: 1]
11	0	return true;
12	2	
13	2	return false;
14	2	}

Coverage Report

Created: 2020-09-02 17:42

Click here for information about interpreting this report.

Filename	Function Coverage	Line Coverage	Region Coverage	Branch Coverage
scratch/aphipps/llvmtest/cov/demo/brdemo.cc	100.00% (2/2)	96.15% (25/26)	90.00% (9/10)	83.33% (5/6)
Totals	100.00% (2/2)	96.15% (25/26)	90.00% (9/10)	83.33% (5/6)

Generated by llvm-cov -- llvm version 12.0.0git

What is Branch Coverage?

- **Condition**
  - A leaf-level boolean expression (cannot be broken down into simpler boolean exprs)
    - if (x == 2) …
    - A condition yields a Branch that evaluates to either *true* or *false*
- **Decision**
  - A boolean expression composed of conditions and zero or more logical operators
    - if ((x == 2) && (y == 4)) …
  - A *decision* without a logical operator is a *condition*

- **LLVM Branch Coverage** provides a measurement of Condition *outcomes*. i.e. whether *conditions* evaluate to both *true* and *false*

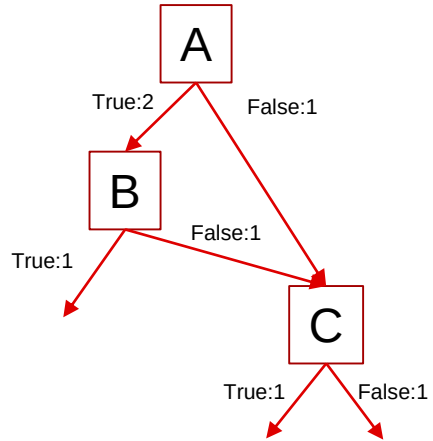
```
  9|      2|bool foo (int x, int y) {
 10|      2|  if ((x > 0) && (y > 0))
------------------
|  Branch (10:7): [True: 1, False: 1]
|  Branch (10:18): [True: 0, False: 1]
------------------
 11|      0|    return true;
 12|      2|
 13|      2|  return false;
 14|      2|}
```

The Limits of Branch Coverage

```
bool test(bool A, bool B, bool C, bool D) {
  return (A && B) || C;
}
```

- When testing, how can we know that we've covered all critical paths?

'A'	'B'	'C'	Result
F	-	F	F
F	-	T	T
T	T	-	T
T	F	T	T
T	F	F	F



- 100% Branch Coverage leaves out critical paths

What is MC/DC?

- “Modified Condition/Decision Coverage”
- A metric pertaining to *conditions* in a Boolean expression *decision* in which:
  - Each condition has been shown to affect that decision outcome *independently*
- “Modified” refers to changing the test input to yield a different test path
  - Given ***n*** conditions, there are $\boldsymbol{2^n}$ total possible test paths (exponential)
  - Only ***n+1*** test paths required to show MC/DC (linear)

Example

- `return ((A && B) || C);`

Test Vector	'A'	'B'	'C'	Result
1	F	-	F	F
2	F	-	T	T
3	T	T	-	T
4	T	F	T	T
5	T	F	F	F

- MC/DC is achieved if an "Independence Pair" can be found for each condition
  - As the condition outcome is varied between True/False, the Result also varies True/False
  - All other condition outcomes are held *fixed* or *don't-care (unevaluatable/masked out)*

Example

- `return ((A && B) || C);`

Test Vector	'A'	'B'	'C'	Result	'A' Pair
1	F	-	F	F	*
2	F	-	T	T	
3	T	T	-	T	*
4	T	F	T	T	
5	T	F	F	F	

- MC/DC is achieved if an "Independence Pair" can be found for each condition
  - As the condition outcome is varied between True/False, the Result also varies True/False
  - All other condition outcomes are held *fixed* or *don't-care (unevaluatable/masked out)*

Example

- `return ((A && B) || C);`

Test Vector	'A'	'B'	'C'	Result	'A' Pair	'B' Pair
1	F	-	F	F	*	
2	F	-	T	T		
3	T	T	-	T	*	*
4	T	F	T	T		
5	T	F	F	F		*

- MC/DC is achieved if an "Independence Pair" can be found for each condition
  - As the condition outcome is varied between True/False, the Result also varies True/False
  - All other condition outcomes are held *fixed* or *don't-care (unevaluatable/masked out)*

Example

- `return ((A && B) || C);`

Test Vector	'A'	'B'	'C'	Result	'A' Pair	'B' Pair	'C' Pair
1	F	-	F	F	*		*
2	F	-	T	T			*
3	T	T	-	T	*	*	
4	T	F	T	T			
5	T	F	F	F		*	

- MC/DC is achieved if an "Independence Pair" can be found for each condition
  - As the condition outcome is varied between True/False, the Result also varies True/False
  - All other condition outcomes are held *fixed* or *don't-care (unevaluatable/masked out)*

Example

- `return ((A && B) || C);`

Test Vector	'A'	'B'	'C'	Result	'A' Pair	'B' Pair	'C' Pair
1	F	-	F	F	*		*
2	F	-	T	T			*
3	T	T	-	T	*	*	
4	T	F	T	T			(*)
5	T	F	F	F		*	(*)

- MC/DC is achieved if an "Independence Pair" can be found for each condition
  - As the condition outcome is varied between True/False, the Result also varies True/False
  - All other condition outcomes are held *fixed* or *don't-care (unevaluatable/masked out)*

LLVM Coverage Report Visualization + MC/DC

Coverage Report

Created: 2022-10-25 17:17

Click here for information about interpreting this report.

Filename	Function Coverage	Line Coverage	Region Coverage	Branch Coverage	MC/DC
scratch/aphipps/llvmtest/cov/mcdc-demo.cpp	100.00% (2/2)	100.00% (10/10)	100.00% (6/6)	83.33% (5/6)	66.67% (2/3)
Totals	**100.00% (2/2)**	**100.00% (10/10)**	**100.00% (6/6)**	**83.33% (5/6)**	**66.67% (2/3)**

Generated by llvm-cov -- llvm version 16.0.0git

LLVM Coverage Visualization + MC/DC

```
Line    Count   Source
   1            #include "mcdc-demo.h"
   2
   3        3   bool test(bool A, bool B, bool C) {
   4        3     return ((A && B) || C);
                  Branch (4:12): [True: 2, False: 1]
                  Branch (4:17): [True: 1, False: 1]
                  Branch (4:23): [True: 0, False: 2]
                  MC/DC Decision Region (4:11) to (4:24)

                  Number of Conditions: 3
                     Condition C1 --> (4:12)
                     Condition C2 --> (4:17)
                     Condition C3 --> (4:23)

                  Executed MC/DC Test Vectors:

                     C1, C2, C3    Result
                  1 { F,  -,  F  = F      }
                  2 { T,  F,  F  = F      }
                  3 { T,  T,  -  = T      }

                  C1-Pair: covered: (1,3)
                  C2-Pair: covered: (2,3)
                  C3-Pair: not covered
                  MC/DC Coverage for Expression: 66.67%
   5        3   }
```

Branch Coverage View

Condition Aliases C{1, 2, 3, …} and Source Location Mapping

Actual Executed Test Vectors

Calculated Metrics for Expression

When is MC/DC really important?

- Required for safety-critical embedded application development
  - *Automotive* (ISO 26262 “Road vehicles – Functional Safety” standard)
  - *Aviation* (DO-178 Aviation standard)
  - Applicable also to *Railway*, *Industrial*, *Medical*, and *Space*

- LLVM is being used increasingly in the embedded space
  - Supporting MC/DC in LLVM makes sense to facilitate embedded development

How is MC/DC implemented in LLVM?

What has been done? What can LLVM do?

- LOG-based MC/DC (most common and very robust)
  - Code is instrumented to track condition outcomes of a test vector
  - Data is output to a file (or stdout) during execution and used as input to a tool

- Today, LLVM really only has raw profile counter data in memory
  - Counters really aren't suitable to tracking test vector execution, and they don't scale well

- **Goal**: Support LOG-based MC/DC without outputting data to a file
  - Leverage clang-based instrumentation to track condition outcomes
  - Efficiently store the data to memory where it can be extracted by coverage tools

Design Concept: Bitmap Coverage Objects

- Track a **global bitmap** in memory in which each bit represents an executed test vector
  - Instrumented per Boolean expression with two or more conditions
    - Treated *like* profile counters but handled differently and kept in a separate section in memory
  - Variable-length, depending on number of possible test vectors (between **8 bits** and **2^n bits**)
  - 2-3 conditions:
    - (A && B) || C → **8-bits**
  - 4 conditions:
    - (A && B) || (C && D) → **16-bits**
  - 5 conditions:
    - (A && B) || (C && D) || E → **32-bits**

- We'll limit the number of conditions measured to **six** to keep everything small

How do Bitmap bits map to test vectors?

- if ((A && B) || (C && D)) …

Test Vector	'A'	'B'	'C'	'D'
1	T	T	-	-
2	T	F	T	T
3	T	F	T	F
4	T	F	F	-
5	F	-	T	T
6	F	-	T	F
7	F	-	F	-

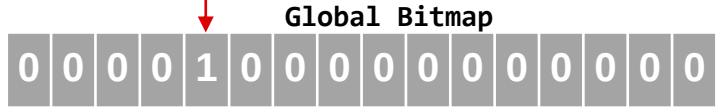
Test Vector	'A'	'B'	'C'	'D'	Value
1	1	1	0	0	12
2	1	0	1	1	11
3	1	0	1	0	10
4	1	0	0	0	8
5	0	0	1	1	3
6	0	0	1	0	2
7	0	0	0	0	0

- Goal: instrument the evaluation of each condition in a Boolean expression
  - The resulting ***value*** gives us an ***index*** into a global, Decision-level Bitmap

How do Bitmap bits map to test vectors?

- The ***value*** of each test vector execution yields an ***index*** into a Global bitmap
  - i.e. Each bit in the global mask represents a test vector
- if ((A && B) || (C && D)) …

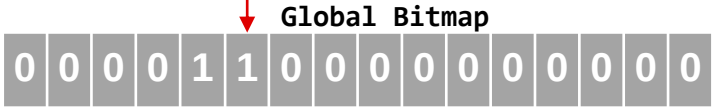
Test Vector	'A'	'B'	'C'	'D'	Value
1	1	1	0	0	12
2	1	0	1	1	11
3	1	0	1	0	10
4	1	0	0	0	8
5	0	0	1	1	3
6	0	0	1	0	2
7	0	0	0	0	0



How do Bitmap bits map to test vectors?

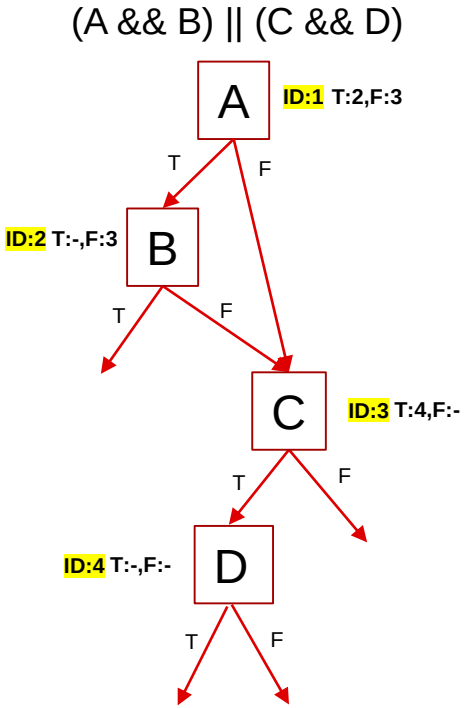
- The ***value*** of each test vector execution yields an ***index*** into a Global Bitmap
  - i.e. Each bit in the global mask represents a test vector
- if ((A && B) || (C && D)) …

Test Vector	'A'	'B'	'C'	'D'	Value
1	1	1	0	0	12
2	1	0	1	1	11
3	1	0	1	0	10
4	1	0	0	0	8
5	0	0	1	1	3
6	0	0	1	0	2
7	0	0	0	0	0



Source Region Mapping

- Introduce a new Decision mapping region type
  - Ties a Boolean expression to source code
  - Contains index of a Decision-level Bitmap
  - Contains number of conditions
- Extend existing Branch Regions to include IDs
  - Represents Control Flow through conditions
- Keep language-specific information away from llvm-cov!
  - Only give it what it needs to do the MC/DC analysis
  - IDs allow llvm-cov to construct list of possible test vectors



Visualization (llvm-cov)

- Extract the Branch Region IDs and build the list of possible test vectors

```
Function (foo)
-    CodeRegion1 (9:24-10:23)
-    CodeRegion2 (11:0-11:12)
-    CodeRegion3 (12:0-14:0)
MCDCDecisionRegion:
-    MCDCDecisionRegion (10:4-10:23)
MCDCBranchRegions:
-    MCDCBranchRegion1 (10:5-10:11)
-    MCDCBranchRegion2 (10:16-10:22)
-    MCDCBranchRegion3 (10:27-10:33)
-    MCDCBranchRegion4 (10:38-10:44)
```

Test Vector	'A'	'B'	'C'	'D'	Value
1	1	1	0	0	12
2	1	0	1	1	11
3	1	0	1	0	10
4	1	0	0	0	8
5	0	0	1	1	3
6	0	0	1	0	2
7	0	0	0	0	0

Visualization (llvm-cov)

- Construct the list of executed test vectors

Test Vector	'A'	'B'	'C'	'D'	Value
1	1	1	0	0	12
2	1	0	1	1	11
3	1	0	1	0	10
4	1	0	0	0	8
5	0	0	1	1	3
6	0	0	1	0	2
7	0	0	0	0	0

```
Function (foo)
-    CodeRegion1 (9:24-10:23)
-    CodeRegion2 (11:0-11:12)
-    CodeRegion3 (12:0-14:0)
MCDCDecisionRegion:
-    MCDCDecisionRegion (10:4-10:23)
MCDCBranchRegions:
-    MCDCBranchRegion1 (10:5-10:11)
-    MCDCBranchRegion2 (10:16-10:22)
-    MCDCBranchRegion3 (10:27-10:33)
-    MCDCBranchRegion4 (10:38-10:44)
```

Visualization (llvm-cov)

- Look for an **Independence Pair** for each Condition

Test Vector	‘A’	‘B’	‘C’	‘D’	Result
2	T	F	T	T	T
3	T	F	T	F	F
5	F	-	T	T	T

Visualization (llvm-cov)

- Look for an **Independence Pair** for each Condition

Test Vector	‘A’	‘B’	‘C’	‘D’	Result	A-Pair
2	T	F	T	T	T	-
3	T	F	T	F	F	-
5	F	-	T	T	T	-

- Condition A: No Independence Pair Found

Visualization (llvm-cov)

- Look for an **Independence Pair** for each Condition

Test Vector	‘A’	‘B’	‘C’	‘D’	Result	A-Pair	B-Pair
2	T	F	T	T	T	-	-
3	T	F	T	F	F	-	-
5	F	-	T	T	T	-	-

- Condition B: No Independence Pair Found

Visualization (llvm-cov)

- Look for an **Independence Pair** for each Condition

Test Vector	‘A’	‘B’	‘C’	‘D’	Result	A-Pair	B-Pair	C-Pair
2	T	F	T	T	T	-	-	-
3	T	F	T	F	F	-	-	-
5	F	-	T	T	T	-	-	-

- Condition C: No Independence Pair Found

Visualization (llvm-cov)

- Look for an **Independence Pair** for each Condition

Test Vector	'A'	'B'	'C'	'D'	Result	A-Pair	B-Pair	C-Pair	D-Pair
2	T	F	T	T	T	-	-	-	*
3	T	F	T	F	F	-	-	-	*
5	F	-	T	T	T	-	-	-	

- Condition D: Independence Pair Found!

  - A-Pair: not covered
  - B-Pair: not covered
  - C-Pair: not covered
  - D-Pair: covered

- MC/DC Coverage for Expression: 25%

Current State of LLVM MC/DC

- Implementation is complete -- in the process of upstreaming the work!
  - Phabricator Review https://reviews.llvm.org/D136385
- Will be included with stock LLVM Source-based Code Coverage
  - But enabled in clang via command-line option
  - `clang -fprofile-instr-generate -fcoverage-mapping -fmcdc foo.cc -o foo`
- A lot of ways to improve MC/DC and Branch Coverage! Want to be involved?
  - Contact me! a-phipps@ti.com
- 2020 Branch Coverage Presentation
  - https://www.youtube.com/watch?v=H1hvtJPGWNQ

Thank you!

- Acknowledgements
  - Vedant Kumar, Apple