AddressSanitizer
+
Code Coverage

Kostya Serebryany, Google

EuroLLVM 2014

New and shiny -fprofile-instr-generate

- Coming this year
- Fast BB-level code coverage
- Increment a counter per every (*) BB
  - Possible contention on counters
- Creates special non-code sections
  - Counters
  - Function names, line numbers

Meanwhile: ASanCoverage

- Tiny prototype-ish thing:
  - Part of AddressSanitizer
  - 30 lines in LLVM, 100 in run-time

- Function- or BB- level coverage
  - Booleans only, not counters
  - No contention
  - No extra sections in the binary

At compile time:

```
if (!*BB_Guard) {
  __sanitizer_cov();
  *BB_Guard = 1;
}
```

At run time

```
void __sanitizer_cov() {
  Record(GET_CALLER_PC());
}
```

At exit time

- For every binary/DSO in the process:
  - Dump observed PCs in a separate file as 4-byte offsets

At analysis time

- Compare/Merge using 20 lines of python
- Symbolize using regular DWARF

```
% cat cov.c
int main() { }
% clang -g -fsanitize=address -mllvm -asan-coverage=1 cov.
c
% ASAN_OPTIONS=coverage=1 ./a.out
% wc -c *sancov
4 a.out.15751.sancov

% sancov.py print a.out.15751.sancov
sancov.py: read 1 PCs from a.out.15751.sancov
sancov.py: 1 files merged; 1 PCs total
0x4850b7

% sancov.py print *.sancov | llvm-symbolizer --obj=a.out
main
/tmp/cov.c:1:0
```

Fuzzing with coverage feedback

- Test corpus: N random tests
- Randomly mutate random test
  - If new BB is covered -- add this test to the corpus
- Many new bugs in well fuzzed projects!

Feedback from our customers

- Speed is paramount
- Binary size is important
  - Permanent & temporary storage, tmps, I/O
  - Stripping non-code section helps partially, but complicates the process
- Booleans per BB is enough

Challenge: Chromium sandbox

- Chromium sandbox forbids open()

Issue: compile time

- ASanCoverage creates too many new BBs
  - 1 file in Chromium takes 30 minutes to build
  - Same issue as with ASan & MSan
  - We hit N^3 in llvm::SpillPlacement
  - Same bug happens just with -O3 on ARM
  - PR17409: volunteers?

ASanCov vs -fprofile-instr-generate

	ASanCov	-fprofile
Ready to use?	**YES**	**NO**
Binary size increase	~ 5%	> 50% (*)
Executable code size increase	~ 5%	~ 3%
Contention on counters	NO	YES
Output per BB	Boolean	Counter
Debug info	DWARF	Separate
Typical slowdown	< 20%	< 20%

Can we improve -fprofile-instr-generate based on experience with ASanCoverage?

More on counter contention

- Counters are incremented every time the program enters a BB
- Counters are global variables
- Typically no trouble, but…
- Example: multi-threaded codec: same functions are running from N threads
  - Cache line ping-pong => 10x slowdown