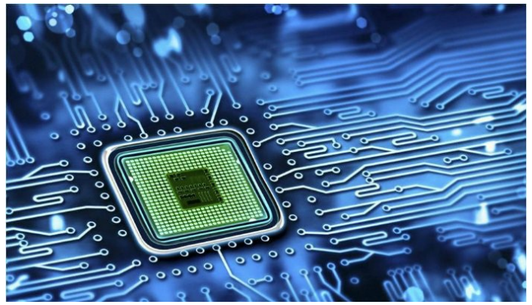
ASSESSING THE EFFECT OF DATA TRANSFORMATIONS ON TEST SUITE COMPILATION

Panagiotis Stratis, **Vanya Yaneva**, Ajitha Rajan

12 October 2018
ESEM'18, Oulu, Finland

THE UNIVERSITY of EDINBURGH
informatics

SOFTWARE IS EVERYWHERE

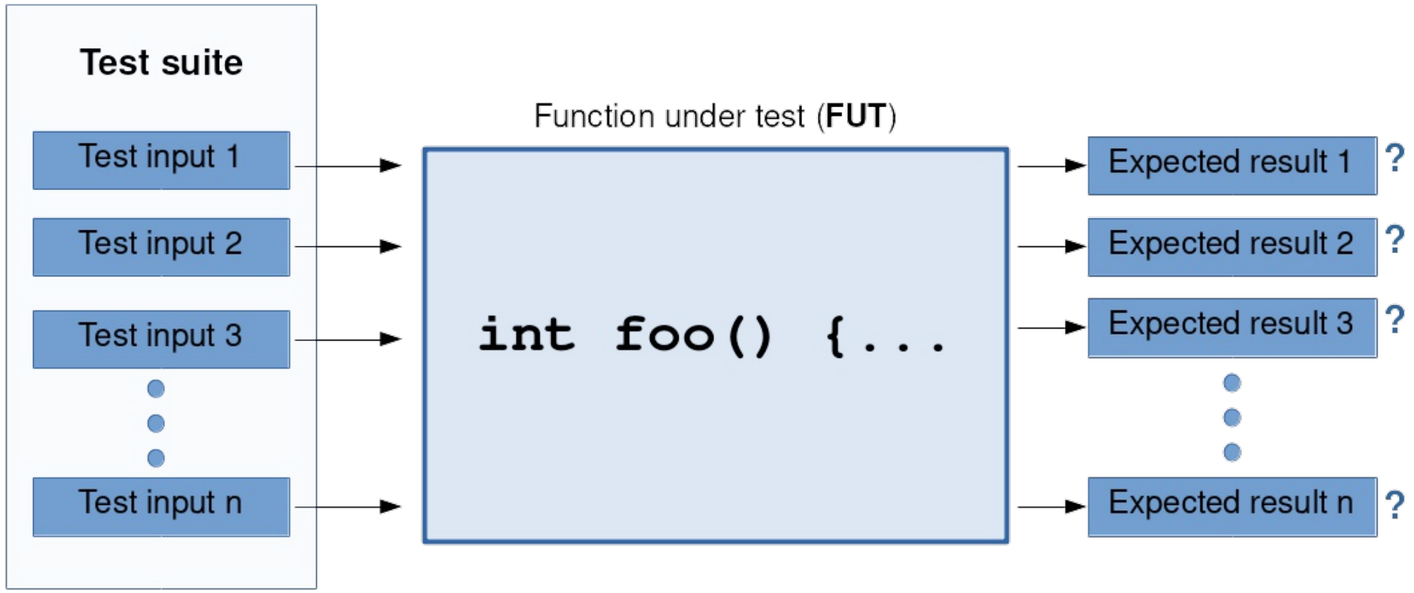




SAFETY AND CORRECTNESS ARE CRUCIAL

TESTING IS CRITICAL

TESTING CAN BE EXTREMELY TIME CONSUMING.

FUNCTIONAL TESTING



HOW DO WE IMPLEMENT TESTING?

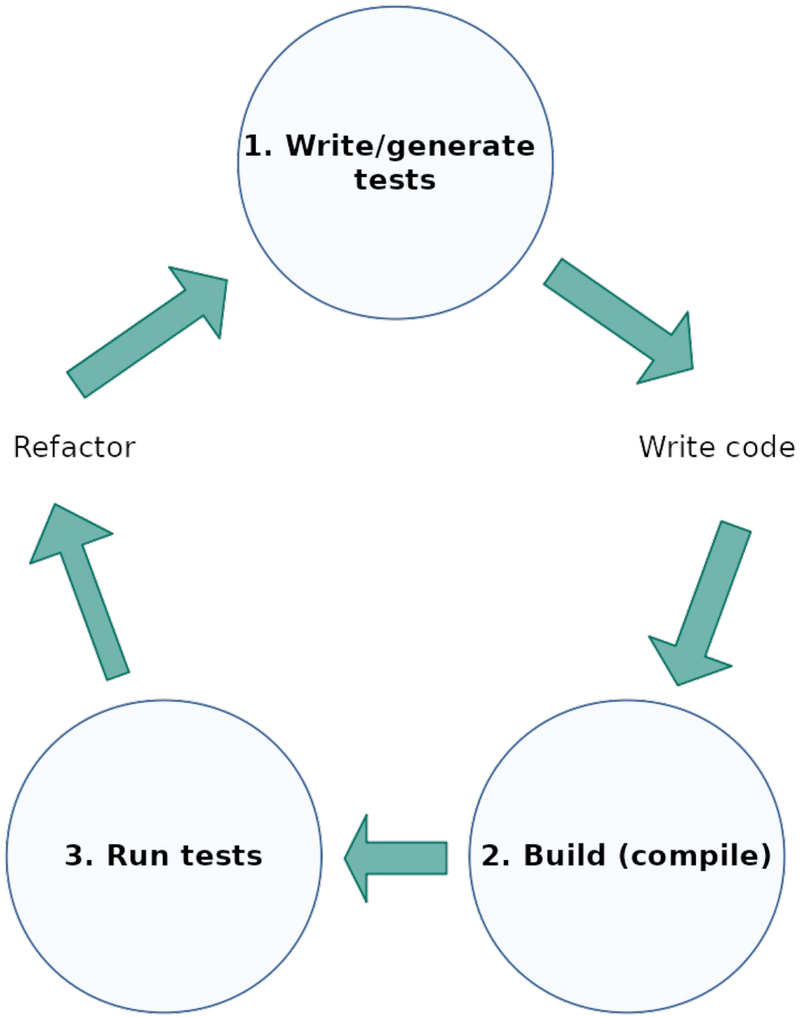
Testing frameworks:

- **GoogleTest** (C++)
- **JUnit** (Java)
- **Mocha** (JavaScript)

- ... and 100s others

WHEN DO WE TEST?

All the time!

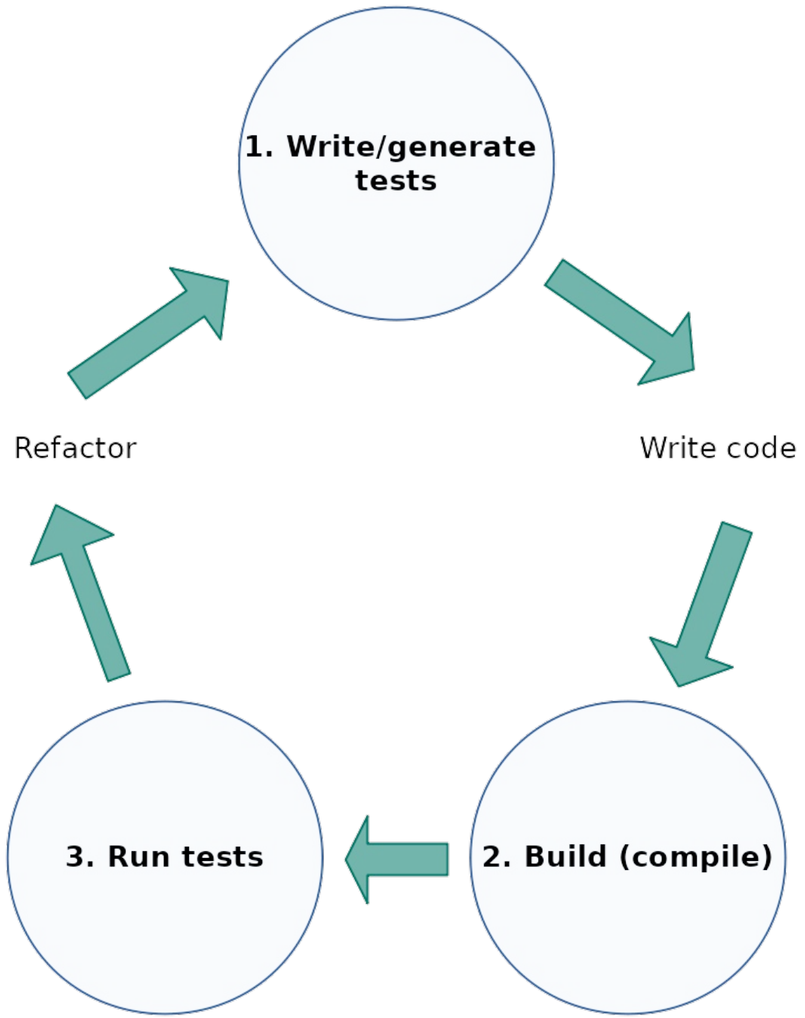


Repeat the cycle:

- every commit
- every merge
  Test-driven development (TDD)
- every day (overnight builds)
  Continuous integration (CI)

Testing takes a significant portion of the development time.

SPEEDING UP TESTING
EXISTING RESEARCH



1. Write/generate tests - ✔
   Automated test generation, parallel test generation
2. Build (compile) - ?
3. Run tests - ✔
   Test suite minimisation, test case prioritisation, parallel test execution

REDUCING COMPILATION TIME IS IMPORTANT

- Large test suites take a long time to compile.
  "... Comparable to running time." [Codeplay Software]
  We need to compile not only the system code, but also the **test code**.

- Compiler optimisations increase compilation time.
  -O1, -O2, -O3

- Test code needs to be compiled often.

CONTRIBUTIONS

- Code transformations ***targeting test code***, resulting in shorter compilation times.
- Empirical evaluation using 15 programs from EEMBC & SPEC and 1 large industry program.
- Speedup in compilation time: **1.3x to 69x**.

OUR APPROACH - EXAMPLE

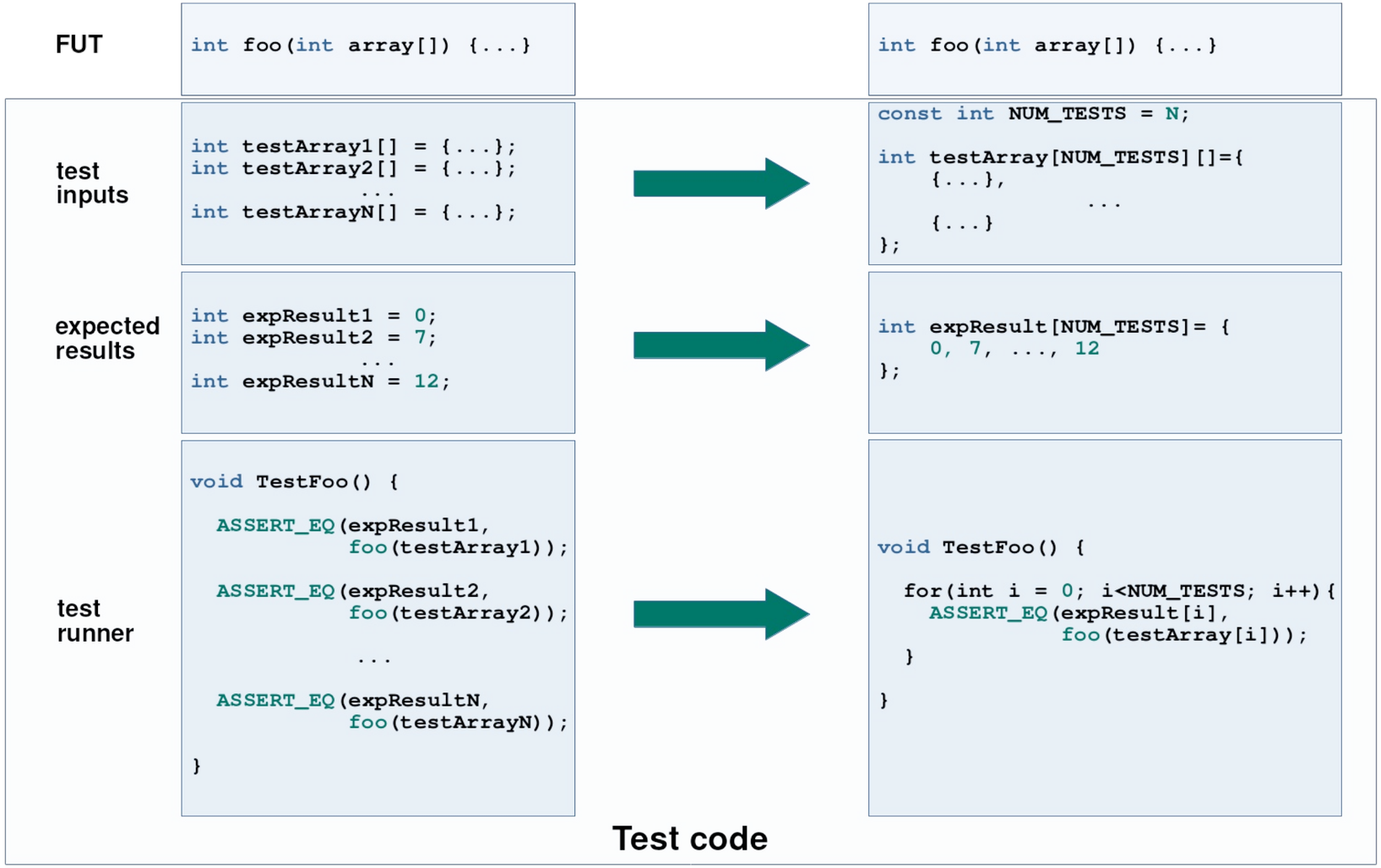
- Contains ***many*** function calls to the FUT - one for each test input.
  - **High compilation overhead** for *function inlining* and *instruction selection*.
- This number grows as more tests are added.

OUR APPROACH - EXAMPLE

- Contains ***many*** function calls to the FUT - one for each test input.
  - **High compilation overhead** for *function inlining* and *instruction selection*.
- This number grows as more tests are added.

Hypothesis:
Reducing the number of calls to the FUT, at compile time, will reduce compilation time.

OUR APPROACH
DATA TRANSFORMATION OF THE TESTS



EVALUATION - RESEARCH QUESTIONS

- RQ1: Compilation speedup
  Does the transformation speedup compilation time?

- RQ2: Scalability
  Does the transformation allow us to compile larger test suites?

- RQ3: Execution time and correctness of testing
  Does the transformation impact the execution time and correctness of testing?

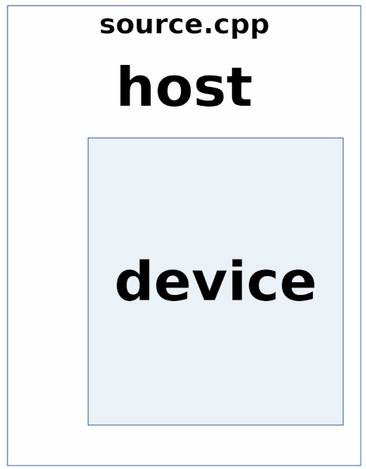
EVALUATION - SUBJECTS

- EEMBC
  - industry-standard embedded systems applications
  - used 5 *telecom* programs & 5 *automotive* programs
- SPEC
  - well-known benchmarks suite of compute intensive applications
  - used 5 programs, including *bzip2, libquantum*
- **Tests:** 10K randomly generated test inputs
- **Compilers:** gcc, clang

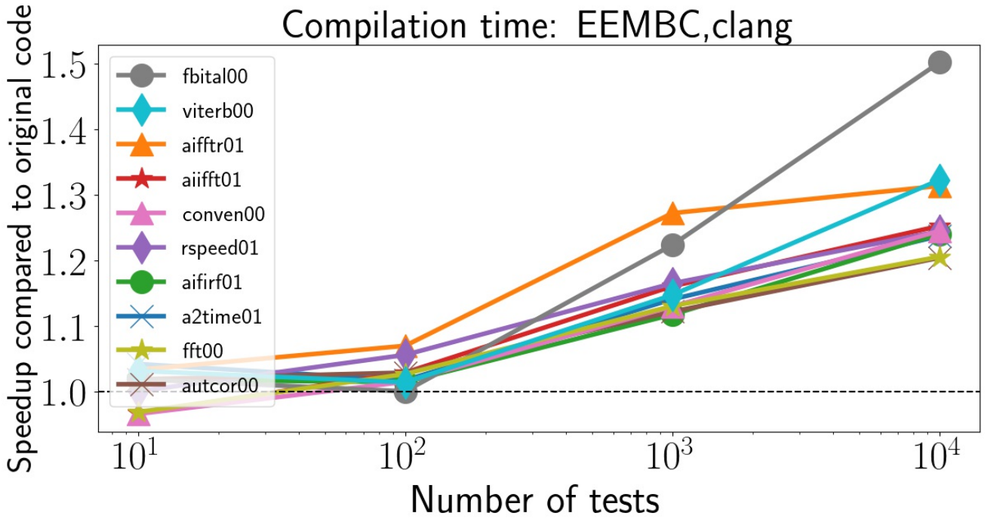
EVALUATION - SUBJECTS

- ComputeCPP,
  Codeplay Software
  implementation of the SYCL
  heterogeneous programming model

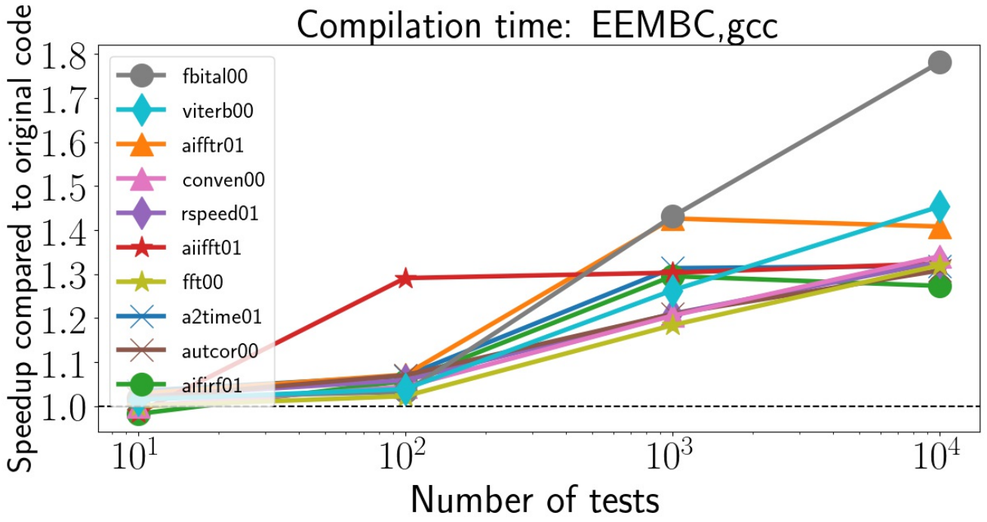
- Tests:
  - 2 test suites - imageTS & bufferTS
  - each has approx. 10K tests, produced by Codeplay developers
- Compilers:
  - 2 custom compilers, based on Clang
  - one for **host** & one for **device**



RQ1: SPEEDUP - EEMBC

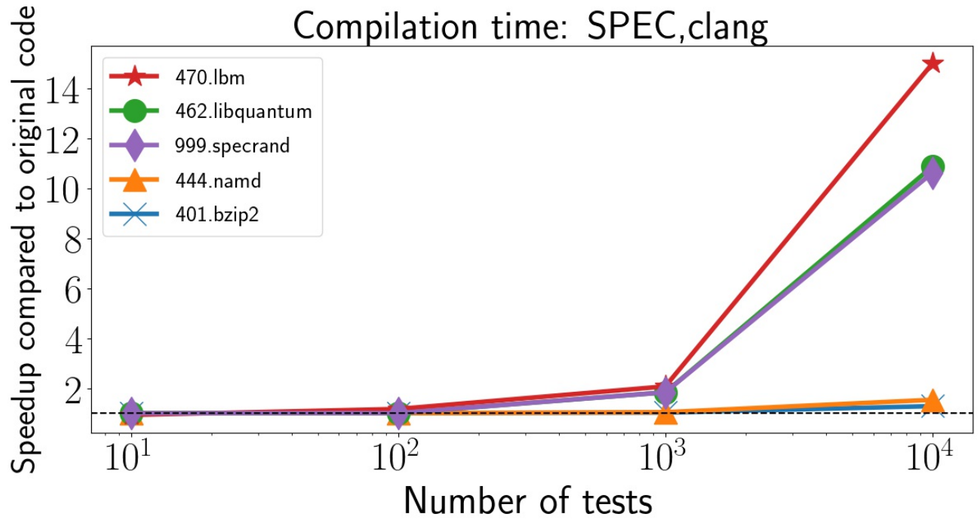


Max speedup **1.5x** (avg. 1.3x)

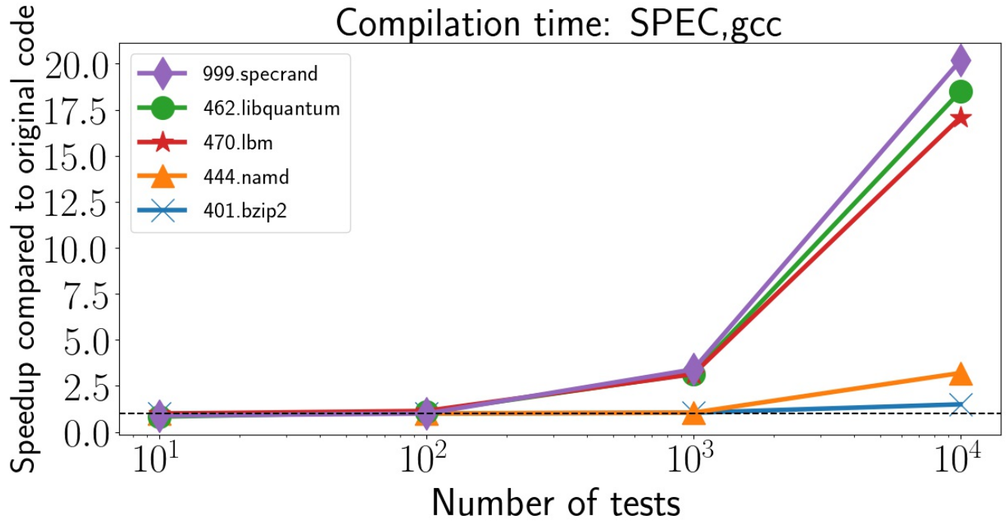


Max speedup **1.8x** (avg. 1.4x)

RQ1: SPEEDUP - SPEC



Max speedup **15x** (avg. 7.9x)



Max speedup **20.2x** (avg. 12x)

SPEEDUP VARIATION ACROSS PROGRAMS

FUT

```
int foo(int array[]) {...}
```

test inputs

```
int testArray1[] = {...};
int testArray2[] = {...};
                ...
int testArrayN[] = {...};
```

expected results

```
int expResult1 = 0;
int expResult2 = 7;
                ...
int expResultN = 12;
```

test runner

```
void TestFoo() {

  ASSERT_EQ(expResult1,
            foo(testArray1));

  ASSERT_EQ(expResult2,
            foo(testArray2));

             ...

  ASSERT_EQ(expResultN,
            foo(testArrayN));

}
```

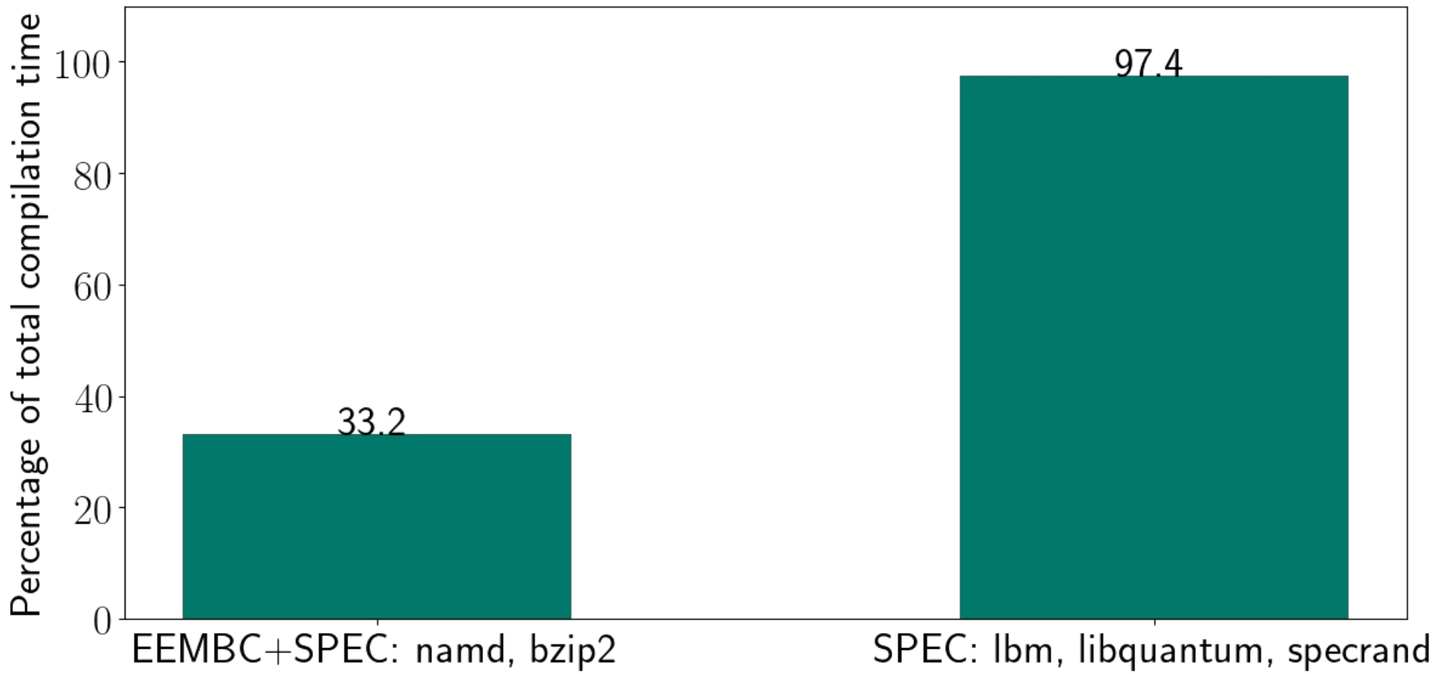
Test code

SPEEDUP VARIATION ACROSS PROGRAMS

- **-ftime-report:** shows time spent compiling individual files and functions.

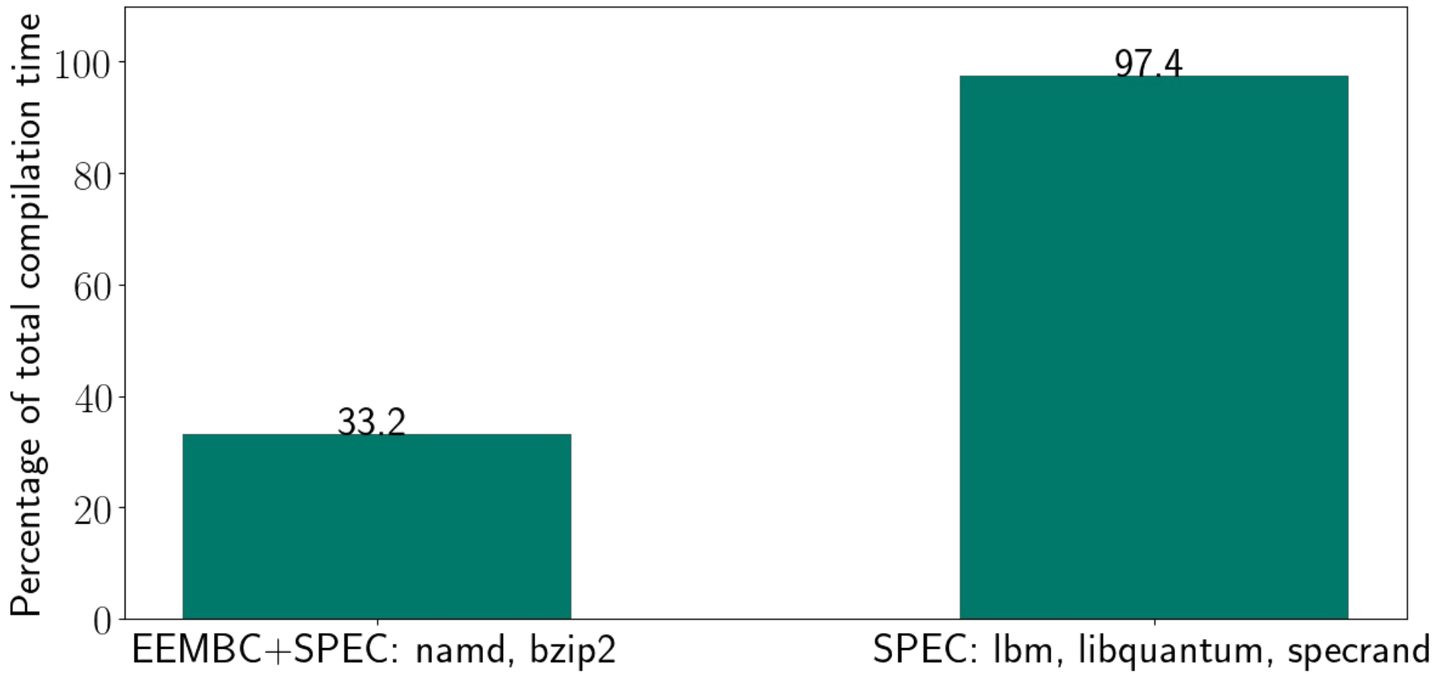
- Compared time to compile **test code** vs whole code.

SPEEDUP VARIATION ACROSS PROGRAMS



Average compilation time of **test code** as percentage of **total** compilation time.

SPEEDUP VARIATION ACROSS PROGRAMS



Average compilation time of **test code** as percentage of **total** compilation time.

Using modular design with pre-compiled libraries greatly improves speedup.

SPEEDUP ANALYSIS

For EEMBC and SPEC,

- Analysed assembly code generated by the compiler.
- Analysed time spent by each compiler pass, using **-ftime-report.**

SPEEDUP ANALYSIS

For EEMBC and SPEC,

- Analysed assembly code generated by the compiler.
- Analysed time spent by each compiler pass, using **-ftime-report.**

Before transformation:

Separate calls to the FUT emitted for **each** test.

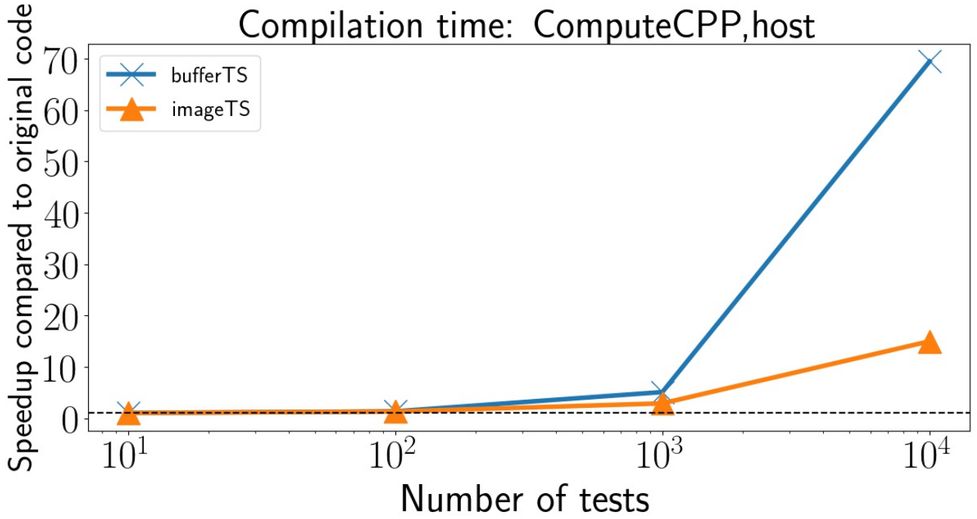
47% of test code compilation spent in
- instruction selection
- function inlining
- combine redundant instructions

After transformation:

1 call to the FUT emitted.

Only **13.6%** of compilation time spent in those passes.

RQ1. SPEEDUP - COMPUTECPP



Max speedup **69.5x** (avg. 42x)

	Orig. time [s]	New time [s]
bufferTS	257	4
imageTS	434	29

RQ2. SCALABILITY

For EEMBC and SPEC,

we generated 10 million random test inputs.

Before transformation:

Clang and gcc ***crash*** when 1 million tests are reached.

After transformation:

Clang and gcc ***successfully*** compile 10 million tests.

for highest optimisation level -O3.

RQ3. EXECUTION TIME AND CORRECTNESS

For EEMBC, SPEC and ComputeCpp,

- **Execution time**
  Our transformation **does not slow down** the execution of the test code.

- **Correctness**
  Our transformation **preserves the correctness** of the test execution.

TAKE AWAY

- Test code can add significant compilation overhead, contributing to total testing time.
- It can be reduced with code transformations on the test code.
- This is particularly effective when using modular design and pre-compiled libraries.