FLiT

Measuring and Locating Floating-Point Variability from Compiler Optimizations

Ignacio Laguna, Harshitha Menon
Lawrence Livermore National Laboratory

Michael Bentley, Ian Briggs, Ganesh Gopalakrishnan
University of Utah

Cindy Rubio González
University of California at Davis

Lawrence Livermore National Laboratory

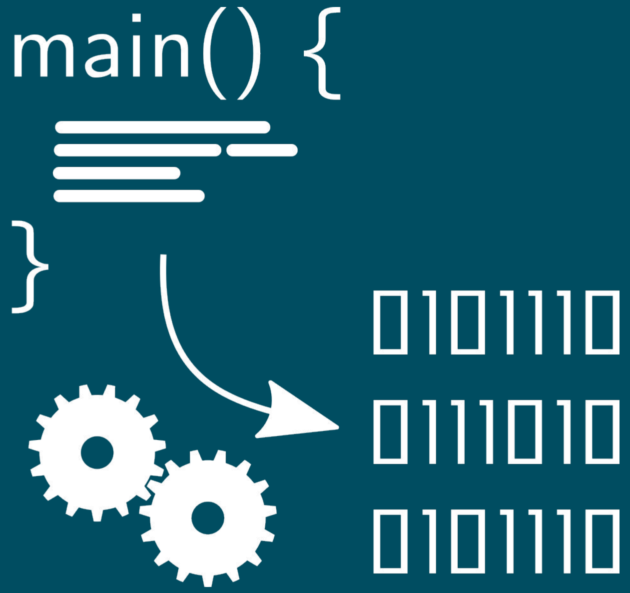
THE UNIVERSITY OF UTAH®

UCDAVIS
UNIVERSITY OF CALIFORNIA

http://fpanalysistools.org/

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344 (LLNL-PRES-780623).

Compilers Can Induce Variability



Compilers have become so stable, we trust them almost implicitly.

I'm here to burst your bubble

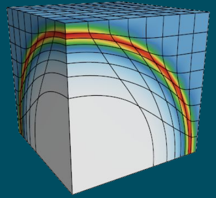
Two different compilations can give vastly different program results

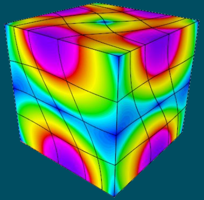
- Not because the compiler has a bug
- Not because the compiler did things wrong
- Not because the compiler doesn't understand

But because the compiler **thinks** you want it

Example of Compiler-Induced Variability

Laghos: A high-order Lagrangian hydrodynamics mini-application





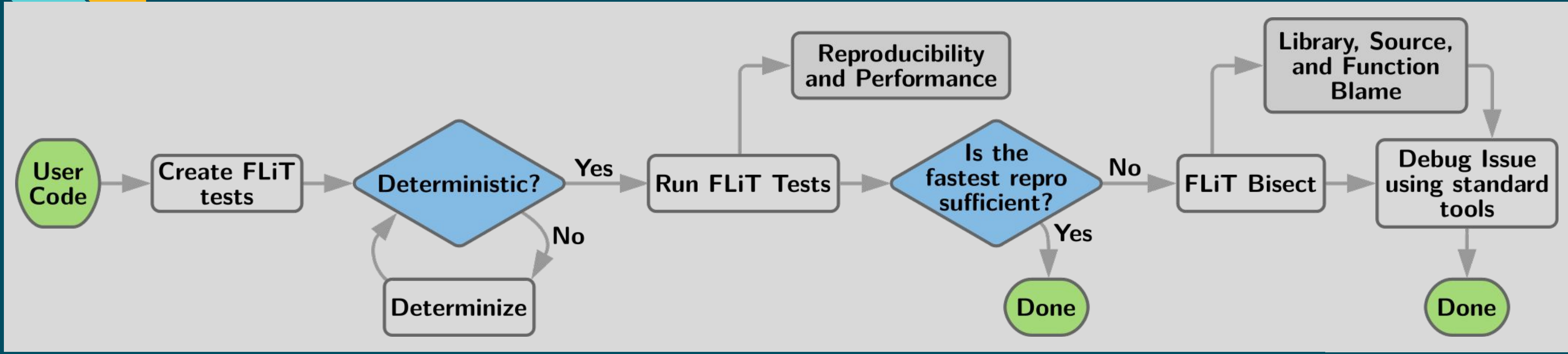
`xlc -O2` → `xlc -O3`

One iteration: **11.2%** relative error!

And speedup by a factor of **2.42**

What happened? How can I investigate it?

FLiT Workflow



Multiple Levels:

1. Determine variability-inducing compilations
2. Analyze the tradeoff of reproducibility and performance
3. Locate variability by identifying files and functions causing variability

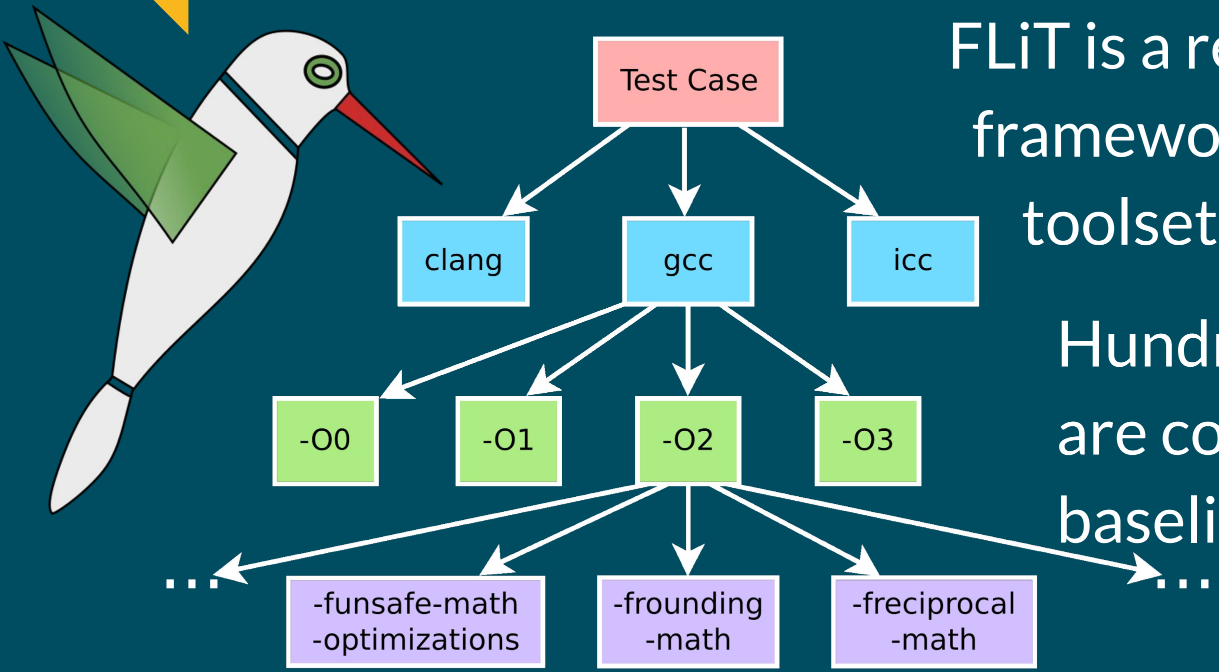
FLiT Installation

FLiT is easy to install

- Very few dependencies
- Use from repository or install on the system

```
 git $ git clone https://github.com/PRUNERS/FLiT.git
Cloning into 'FLiT'...
[...]
 git $ cd FLiT
 FLiT $ make
  src/timeFunction.cpp -> src/timeFunction.o
  src/flitHelpers.cpp -> src/flitHelpers.o
  src/TestBase.cpp -> src/TestBase.o
  src/flit.cpp -> src/flit.o
  src/FlitCsv.cpp -> src/FlitCsv.o
  src/InfoStream.cpp -> src/InfoStream.o
  src/subprocess.cpp -> src/subprocess.o
  src/Variant.cpp -> src/Variant.o
  src/fsutil.cpp -> src/fsutil.o
  mkdir lib
Building lib/libflit.so
 FLiT $ sudo make install
Installing...
  Generating /usr/share/flit/scripts/flitconfig.py
 FLiT $ sudo apt install python3-toml python3-pyelftools
[...]
```

Multi-Compilation Search



FLiT is a reproducibility test framework in the PRUNERS toolset (pruners.github.io).

Hundreds of compilations are compared against a baseline compilation.

Exercises

Exercises with FLiT

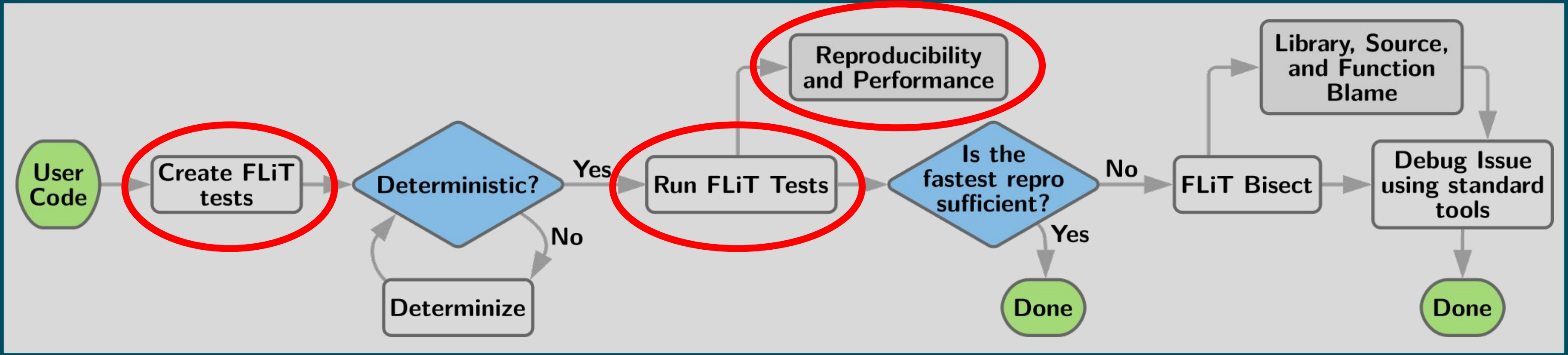
1. MFEM: many compilations and measure variability
2. MFEM: locate site of variability with FLiT Bisect
3. LULESH: auto-run many FLiT Bisects and Bisect-Biggest

Directory Structure

```
Module-FLiT/
├── exercise-1/
├── exercise-2/
├── exercise-3/
├── packages/
├── README.md
└── setup.sh
```

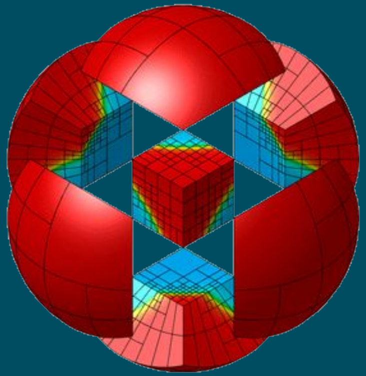
Exercise 1

Exercise 1 - Goal



1. Generate a FLiT test
2. Run the test with many compilations
3. Look at the results

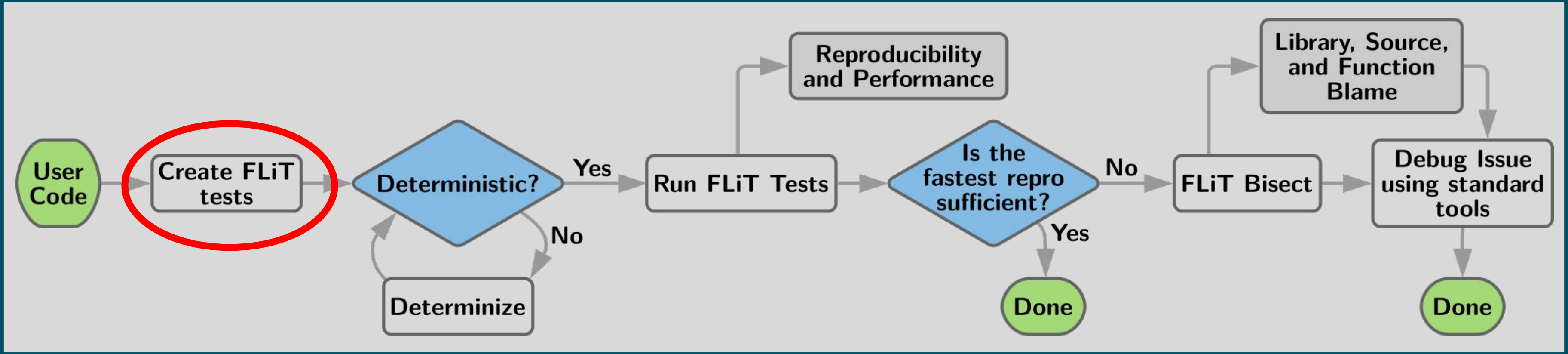
Application: MFEM



source files	97
average functions per file	31
total functions	2,998
source lines of code	103,205

- Open-source finite element library
  - Developed at LLNL
  - https://github.com/mfem/mfem.git
- Provides many example use cases
- Represents real-world code

Exercise 1 - Create MFEM Test



What does it take to create a FLiT test from an MFEM example?

Let’s find out!

Exercise 1 - Create MFEM Test

```
Module-FLiT $ cd exercise-1
```

Let's look at the test for MFEM example #13

`tests/Mfem13.cpp`

```
exercise-1 $ vim tests/MFEM13.cpp
```

or

```
exercise-1 $ pygmentize tests/Mfem13.cpp | cat -n
```

or whatever...

Exercise 1 - Create MFEM Test

tests/MFEM13.cpp

```
// Redefine main() to avoid name clash.  This is the function we will test
#define main mfem_13p_main
#include "ex13p.cpp"
#undef main
// Register it so we can use it in call_main() or call_mpi_main()
FLIT_REGISTER_MAIN(mfem_13p_main);
```

Things to notice:

- Include `ex13p.cpp` from MFEM without modification
- Rename `main()` to `mfem_13p_main()` to avoid name clash
- Register `mfem_13p_main()` with FLiT to be called as a separate process

Exercise 1 - Create MFEM Test

tests/MFEM13.cpp

```
template <typename T>
class Mfem13 : public flit::TestBase<T> {
public:
  Mfem13(std::string id) : flit::TestBase<T>(std::move(id)) {}
  virtual size_t getInputsPerRun() override { return 0; }
  virtual std::vector<T> getDefaultInput() override { return { }; }

  virtual long double compare(const std::vector<std::string> &ground_truth,
                              const std::vector<std::string> &test_results) const override {
    [...]
  }
```

- A simple test setup with no floating-point inputs
- `compare()` does L2 norm and returns % relative difference (skipped)

Exercise 1 - Create MFEM Test

tests/MFEM13.cpp

```cpp
// Only implement the test for double precision
template<>
flit::Variant Mfem13<double>::run_impl(const std::vector<double> &ti) {
  FLIT_UNUSED(ti);

  // Run in a temporary directory so output files don't clash
  std::string start_dir = flit::fsutil::curdir();
  flit::fsutil::TempDir exec_dir;
  flit::fsutil::PushDir pusher(exec_dir.name());
```

- Only double precision is implemented
- Create a temporary directory and go there (for out files)

Exercise 1 - Create MFEM Test

tests/MFEM13.cpp

```
// Run the example's main under MPI
auto meshfile = flit::fsutil::join(start_dir,"data", "beam-tet.mesh");
auto result = flit::call_mpi_main(
                     mfem_13p_main,
                    "mpirun -n 1 --bind-to none",
                    "Mfem13",
                    "--no-visualization --mesh "+ meshfile);
```

- Call `mfem_13p_main()` as a child process with MPI
- Command-line arguments for `mpirun` are given
- For this tutorial, only one MPI process, but can use many
- Command-line arguments for `mfem_13p_main()` are given

Exercise 1 - Create MFEM Test

tests/MFEM13.cpp

```
// Output debugging information
std::ostream &out = flit::info_stream;
out << id << " stdout: " << result.out << "\n";
out << id << " stderr: " << result.err << "\n";
out << id << " return: " << result.ret << "\n";
out.flush();

if (result.ret != 0) {
  throw std::logic_error("Failed to run my main correctly");
}
```

- Result from `call_mpi_main()` have `out`, `err`, and `ret`
- We check for an error using the return code, `ret`

Exercise 1 - Create MFEM Test

tests/MFEM13.cpp

```
// We will be returning a vector of strings that hold the mesh data
std::vector<std::string> retval;
  [...]
// Return the mesh and mode files as strings
return flit::Variant(retval);
```

- We skip the details here
- Return value is a `vector<string>` used by `compare()`

Exercise 1 - Create MFEM Test

```
tests/MFEM13.cpp

116 REGISTER_TYPE(Mfem13)
```

Finally, we register the test class with FLiT

Now, let's look at how the FLiT configuration looks
This has config about compilers and the search space

```
exercise-1 $ vim flit-config.toml
```

Exercise 1 - FLiT Configuration

flit-config.toml

```
[run]
enable_mpi = true
```

- Needed to get the compiler and linker flags for MPI
- Grabs the flags from `mpic++`

Exercise 1 - FLiT Configuration

flit-config.toml

```
[dev_build]
compiler_name = 'g++'
optimization_level = '-O3'
switches = '-mavx2 -mfma'

[ground_truth]
compiler_name = 'g++'
optimization_level = '-O2'
switches = ''
```

Defines the compilations for `make dev` and `make gt`

Exercise 1 - FLiT Configuration

flit-config.toml

```toml
[[compiler]]
binary = 'g++-7'
name = 'g++'
type = 'gcc'
optimization_levels = [
    '-O3',
]
switches_list = [
    '-ffast-math',
    '-funsafe-math-optimizations',
    '-mfma',
]
```

- Defines the “g++” compiler
- Defines the compilation search space

Exercise 1 - FLiT Configuration

flit-config.toml

```
[[compiler]]
binary = 'clang++-6.0'
name = 'clang++'
type = 'clang'
optimization_levels = [
    '-O3',
]
switches_list = [
    '-ffast-math',
    '-funsafe-math-optimizations',
    '-mfma',
]
```

- Defines the "`clang++`" compiler
- Defines the compilation search space

Exercise 1 - Makefile Configuration

A second configuration file: `custom.mk`

- FLiT autogenerates a `Makefile`
- `custom.mk` is included in the `Makefile`
- Tells FLiT how to compile your test(s)

```
exercise-1 $ vim custom.mk
```

Exercise 1 - Makefile Configuration

custom.mk

```
PACKAGES_DIR    := $(abspath ../packages)
MFEM_SRC        := $(PACKAGES_DIR)/mfem
HYPRE_SRC       := $(PACKAGES_DIR)/hypre
METIS_SRC       := $(PACKAGES_DIR)/metis-4.0

SOURCE          :=
SOURCE          += $(wildcard *.cpp)
SOURCE          += $(wildcard tests/*.cpp)

# Compiling all sources of MFEM into the tests takes too long for a tutorial
# skip it.  Instead, we link in the MFEM static library
#SOURCE          += $(wildcard ${MFEM_SRC}/fem/*.cpp)
#SOURCE          += $(wildcard ${MFEM_SRC}/general/*.cpp)
#SOURCE          += $(wildcard ${MFEM_SRC}/linalg/*.cpp)
#SOURCE          += $(wildcard ${MFEM_SRC}/mesh/*.cpp)

# just the one source file to see there is a difference
SOURCE          += ${MFEM_SRC}/linalg/densemat.cpp  # where the bug is
```

Exercise 1 - Makefile Configuration

custom.mk

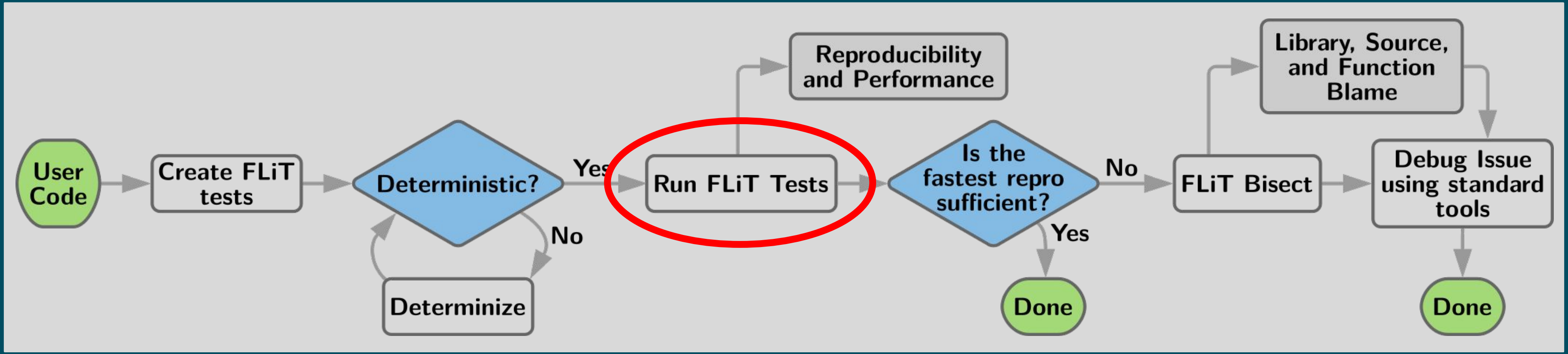
```
CC_REQUIRED     += -I${MFEM_SRC}
CC_REQUIRED     += -I${MFEM_SRC}/examples
CC_REQUIRED     += -isystem ${HYPRE_SRC}/src/hypre/include

LD_REQUIRED     += -L${MFEM_SRC} -lmfem
LD_REQUIRED     += -L${HYPRE_SRC}/src/hypre/lib -lHYPRE
LD_REQUIRED     += -L${METIS_SRC} -lmetis
```

That’s all there is to it

Let’s run it!

Exercise 1 - Run the MFEM Test



Each command has a script.

Run the script or the command from the slide - your choice

Exercise 1 - `./step-01.sh`

```
 exercise-1 $ flit update
Creating ./Makefile
```

- Auto-generate `Makefile`
- Since it is auto-generated, it is usually not committed in a repo

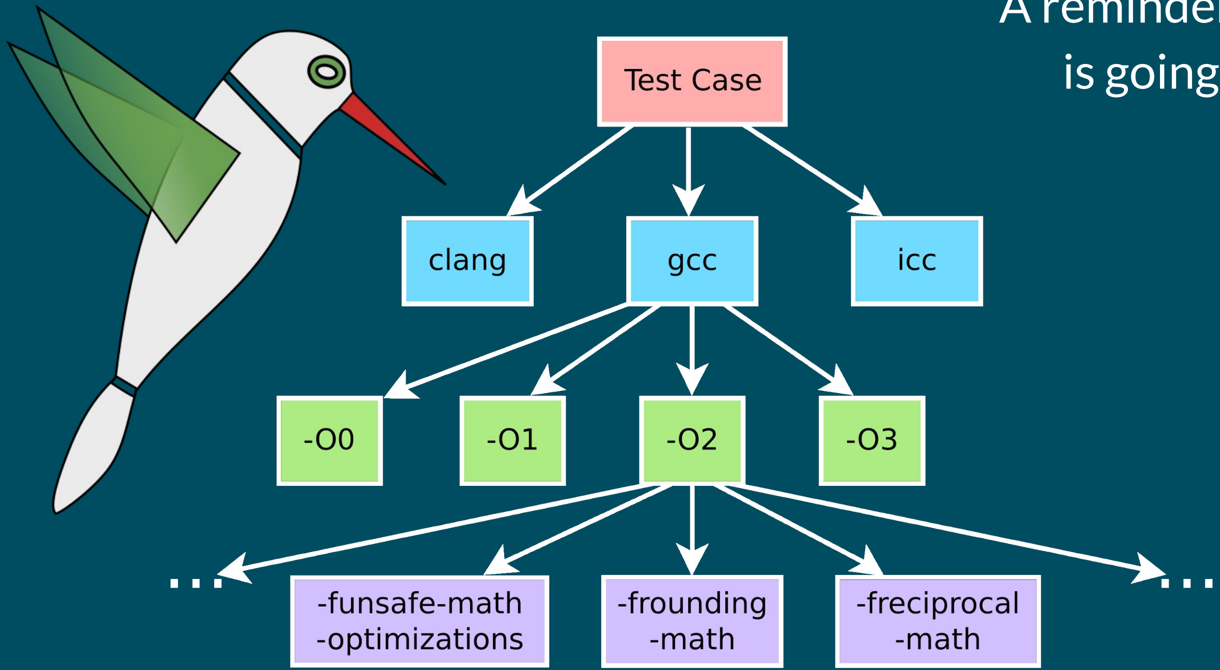
Exercise 1 - `./step-02.sh`

```
exercise-1 $ make runbuild -j1
  mkdir obj/gt
  /home/user1/Module-FLiT/packages/mfem/linalg/densemat.cpp -> obj/gt/densemat.cpp.o
  main.cpp -> obj/gt/main.cpp.o
  tests/Mfem13.cpp -> obj/gt/Mfem13.cpp.o
Building gtrun
  mkdir bin
  mkdir obj/GCC_ip-172-31-8-101_FFAST_MATH_O3
  /home/user1/Module-FLiT/packages/mfem/linalg/densemat.cpp ->
obj/GCC_ip-172-31-8-101_FFAST_MATH_O3/densemat.cpp.o
[...]
```

(takes about 1 minute)

- For verbose output use `make VERBOSE=1 ...`
- Will make all compilations from search space into `bin/`
- Can do more parallelism (but not for this tutorial)

Exercise 1 - `./step-02.sh`



A reminder about what is going on here...

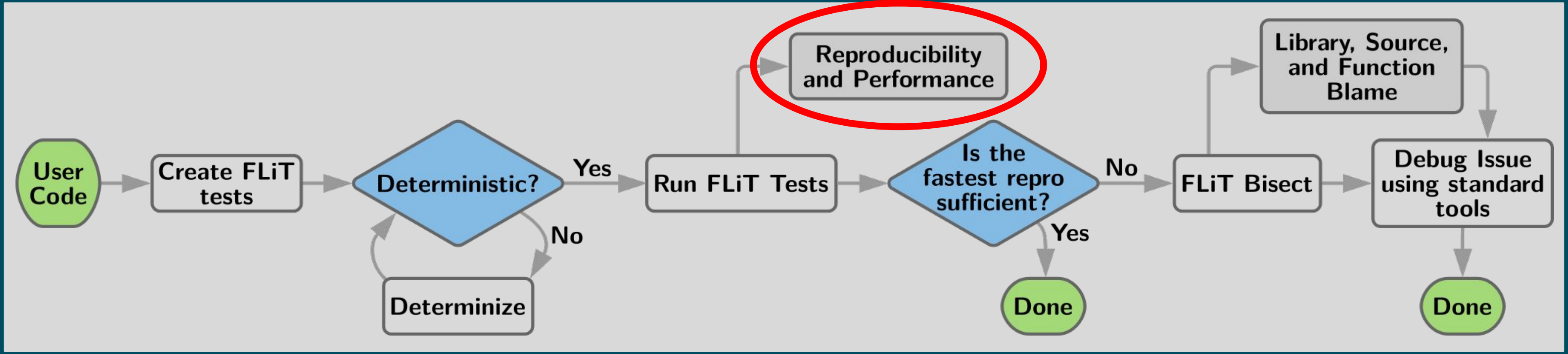
Exercise 1 - `./step-03.sh`

```
exercise-1 $ make run -j1
  mkdir results
  gtrun -> ground-truth.csv
  results/GCC_ip-172-31-8-101_FFAST_MATH_O3-out ->
results/GCC_ip-172-31-8-101_FFAST_MATH_O3-out-comparison.csv
  results/GCC_ip-172-31-8-101_FUNSAFE_MATH_OPTIMIZATIONS_O3-out ->
results/GCC_ip-172-31-8-101_FUNSAFE_MATH_OPTIMIZATIONS_O3-out-comparison.csv
  results/GCC_ip-172-31-8-101_MFMA_O3-out ->
results/GCC_ip-172-31-8-101_MFMA_O3-out-comparison.csv
  results/CLANG_ip-172-31-8-101_FFAST_MATH_O3-out ->
results/CLANG_ip-172-31-8-101_FFAST_MATH_O3-out-comparison.csv
  results/CLANG_ip-172-31-8-101_FUNSAFE_MATH_OPTIMIZATIONS_O3-out ->
results/CLANG_ip-172-31-8-101_FUNSAFE_MATH_OPTIMIZATIONS_O3-out-comparison.csv
  results/CLANG_ip-172-31-8-101_MFMA_O3-out ->
results/CLANG_ip-172-31-8-101_MFMA_O3-out-comparison.csv
  [...]
```

(takes about 1 minute)

- Runs the test and the `compare()` function

Exercise 1 - Analyze Results



Let us look at the generated results

They are in the `results/` directory

Exercise 1 - `./step-04.sh`

```
exercise-1 $ flit import results/*.csv
Creating results.sqlite
Importing results/CLANG_yoga-manjaro_FFAST_MATH_O3-out-comparison.csv
Importing results/CLANG_yoga-manjaro_FUNSAFE_MATH_OPTIMIZATIONS_O3-out-comparison.csv
Importing results/CLANG_yoga-manjaro_MFMA_O3-out-comparison.csv
Importing results/GCC_yoga-manjaro_FFAST_MATH_O3-out-comparison.csv
Importing results/GCC_yoga-manjaro_FUNSAFE_MATH_OPTIMIZATIONS_O3-out-comparison.csv
Importing results/GCC_yoga-manjaro_MFMA_O3-out-comparison.csv
```

Creates `results.sqlite`

Exercise 1 - `./step-05.sh`

```
 exercise-1 $ sqlite3 results.sqlite
SQLite version 3.28.0 2019-04-16 19:49:53
Enter ".help" for usage hints.
sqlite> .tables
runs   tests
sqlite> .headers on
sqlite> .mode column
sqlite> select * from runs;
id          rdate                       label
----------  --------------------------  ------------------
1           2019-07-08 23:05:19.358055  First FLiT Results
```

Two tables in the database:

1. `runs`: has our label and the date and time of importing
2. `tests`: test results with timing

Exercise 1 - `./step-06.sh`

```
sqlite> select compiler, optl, switches, comparison, nanosec from tests;
compiler      optl        switches      comparison  nanosec
------------  ----------  ------------  ----------  ----------
clang++-6.0   -O3         -ffast-math   0.0         2857386994
clang++-6.0   -O3         -funsafe-ma   0.0         2853588952
clang++-6.0   -O3         -mfma         0.0         2858789982
g++-7         -O3         -ffast-math   0.0         2841191528
g++-7         -O3         -funsafe-ma   0.0         2868636192
g++-7         -O3         -mfma         193.007351  2797305220
sqlite> .q
```

One compilation had 193% relative error!

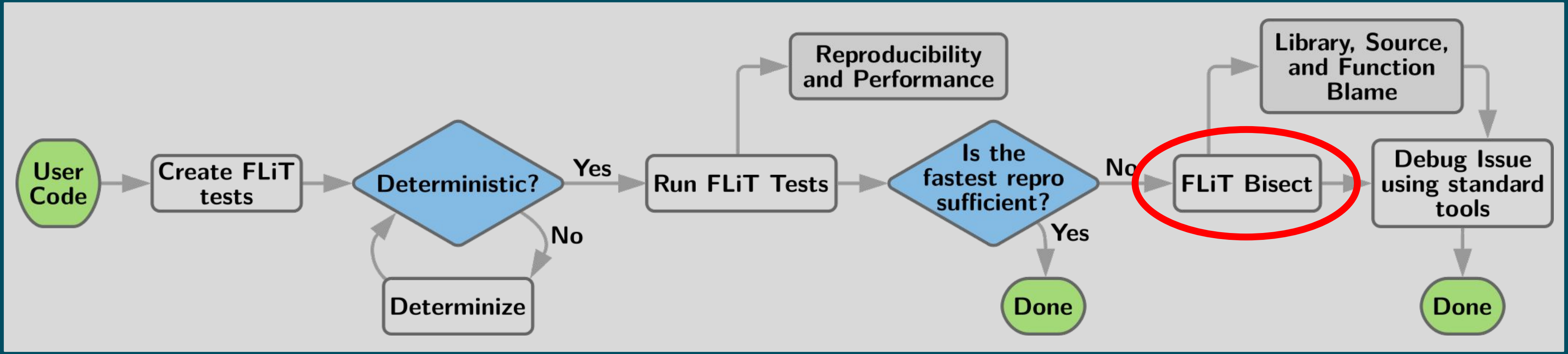
The others had no error.

Now to find the sites in the source code

Exercise 2

```
exercise-1 $ cd ../exercise-2
```

Exercise 2 - FLiT Bisect



We want to find the file(s)/function(s) where FMA caused 193% relative error

Compilation: `g++-7 -O3 -mfma`

Exercise 2 - `./step-07.sh`

What's Different?

```
exercise-2 $ diff -u ../exercise-1/custom.mk ./custom.mk
--- ../exercise-1/custom.mk 2019-07-01 16:09:39.239923037 -0600
+++ custom.mk 2019-07-01 16:07:41.090571010 -0600
@@ -17,9 +17,15 @@
 #SOURCE         += $(wildcard ${MFEM_SRC}/linalg/*.cpp)
 #SOURCE         += $(wildcard ${MFEM_SRC}/mesh/*.cpp)

-# just the one source file to see there is a difference
 SOURCE         += ${MFEM_SRC}/linalg/densemat.cpp  # where the bug is

+# a few more files to make the search space a bit more interesting
+SOURCE         += ${MFEM_SRC}/linalg/matrix.cpp
+SOURCE         += ${MFEM_SRC}/fem/gridfunc.cpp
+SOURCE         += ${MFEM_SRC}/fem/linearform.cpp
+SOURCE         += ${MFEM_SRC}/mesh/point.cpp
+SOURCE         += ${MFEM_SRC}/mesh/quadrilateral.cpp
+
 CC_REQUIRED    += -I${MFEM_SRC}
 CC_REQUIRED    += -I${MFEM_SRC}/examples
 CC_REQUIRED    += -isystem ${HYPRE_SRC}/src/hypre/include
```

Exercise 2 - `./step-08.sh`

Again, we need to regenerate the `Makefile`

```
exercise-2 $ flit update
Creating ./Makefile
```

Before we bisect, remember which compilation caused a problem:

`g++-7 -O3 -mfma`

Exercise 2 - `./step-09.sh`

```
 exercise-2 $ flit bisect --precision=double “g++-7 -O3 -mfma” Mfem13
Updating ground-truth results - ground-truth.csv - done
Searching for differing source files:
  Created ./bisect-04/bisect-make-01.mk - compiling and running - score 193.00735125466363
  Created ./bisect-04/bisect-make-02.mk - compiling and running - score 193.00735125466363
  Created ./bisect-04/bisect-make-03.mk - compiling and running - score 0.0
  Created ./bisect-04/bisect-make-04.mk - compiling and running - score 193.00735125466363
    Found differing source file /home/user1/Module-FLiT/packages/mfem/linalg/densemat.cpp: score
193.00735125466363
[...]
All variability inducing symbols:
  /home/user1/Module-FLiT/packages/mfem/linalg/ densemat.cpp:3692
_ZN4mfem13AddMult_a_AAtEdRKNS_11DenseMatrixERS0_ --  mfem::AddMult_a_AAt(double, mfem::DenseMatrix
const&, mfem::DenseMatrix&) (score 193.00735125466363)
```

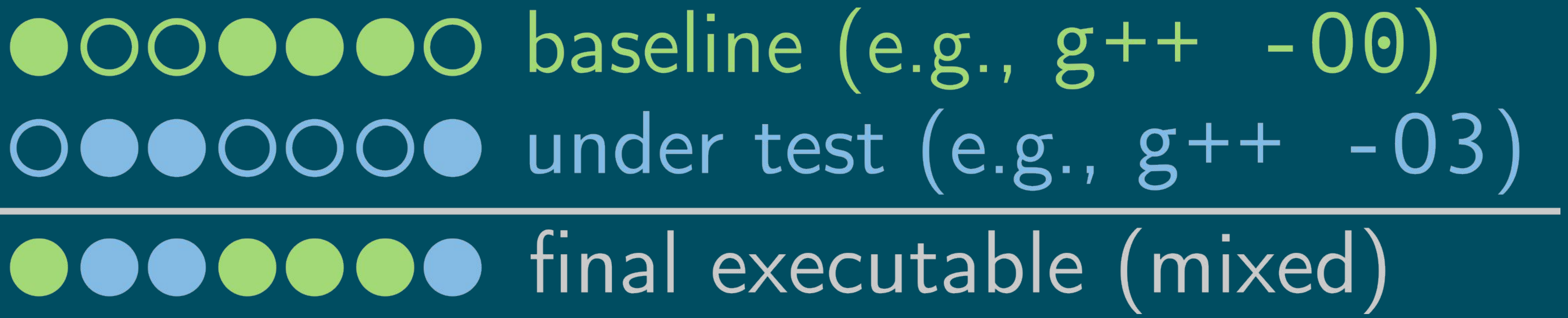
(takes approximately 1 minute 30 seconds)

- Finds the file: `densemat.cpp`
- Finds the function: `mfem::AddMult_a_AAt()`

Exercise 2 - Bisect Details

First locate variability files

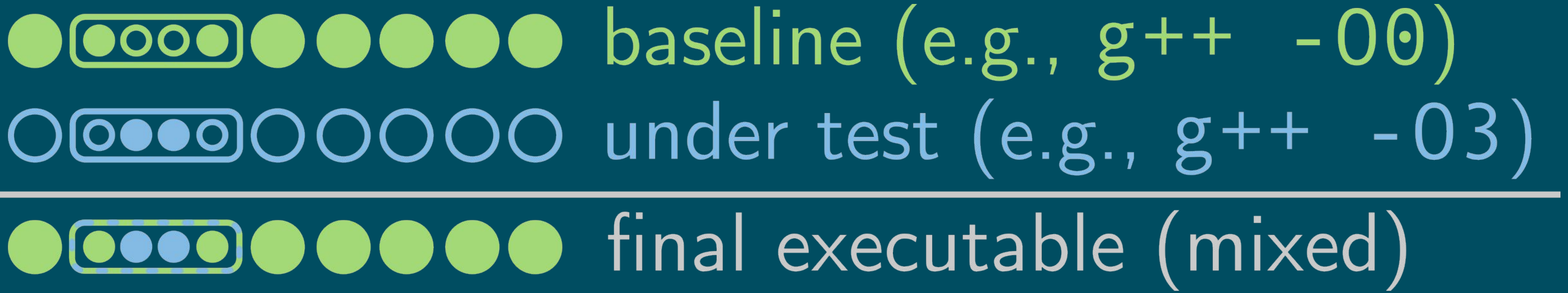
Approach: combine object files from the two compilations



Exercise 2 - Bisect Details

Approach: combine symbols **after compilation**

Convert function symbols into weak symbols



Downside: Requires recompiling with `-fPIC`

Exercise 2 - `./step-10.sh`

```
exercise-2 $ cat -n ../packages/mfem/linalg/densemat.cpp | tail -n +3688 | head -n 24
void AddMult_a_AAt(double a, const DenseMatrix &A, DenseMatrix &AAt)
{
  double d;

  for (int i = 0; i < A.Height(); i++)
  {
     for (int j = 0; j < i; j++)
     {
        d = 0.;
        for (int k = 0; k < A.Width(); k++)
        {
           d += A(i,k) * A(j,k);
        }
        AAt(i, j) += (d *= a);
        AAt(j, i) += d;
     }
     d = 0.;
     for (int k = 0; k < A.Width(); k++)
     {
        d += A(i,k) * A(i,k);
     }
     AAt(i, i) += a * d;
  }
}
```

Computes

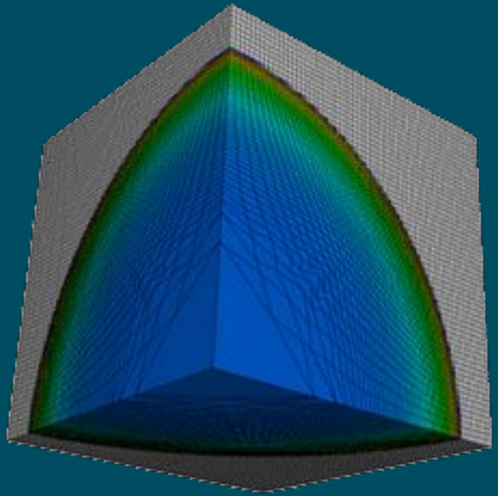
$$M = M + aAA^\top$$

Exercise 3

```
exercise-2 $ cd ../exercise-3
```

Exercise 3 Application: LULESH

- Proxy application developed at LLNL
- Models a shock hydrodynamics problem



Goal: explore more FLiT Bisect functionality

- Auto-Bisect all from `results.sqlite`
- Bisect-Biggest instead of Bisect-All

Exercise 3 - `./step-11.sh`

```
 exercise-3 $ sqlite3 results.sqlite
SQLite version 3.22.0 2018-01-22 18:45:57
Enter ".help" for usage hints.
sqlite> .headers on
sqlite> .mode column
sqlite> select compiler, optl, switches, comparison, nanosec from tests;
compiler      optl        switches             comparison            nanosec
------------  ----------  -------------------  --------------------  ----------
clang++-6.0   -O3         -freciprocal-math    5.52511478433538e-05  432218541
clang++-6.0   -O3         -funsafe-math-opt    5.52511478433538e-05  432185456
clang++-6.0   -O3                              0.0                   433397072
g++-7         -O3         -freciprocal-math    5.52511478433538e-05  441362811
g++-7         -O3         -funsafe-math-opt    7.02432004920159      436202864
g++-7         -O3         -mavx2 -mfma         1.02330009691563      416599918
g++-7         -O3                              0.0                   432654778
sqlite> .q
```

Five variability compilations.

Let’s investigate!

Exercise 3 - `./step-12.sh`

```
exercise-3 $ flit update
Creating ./Makefile
```

Nothing surprising here...

Exercise 3 - `./step-13.sh`

```
 exercise-3 $ flit bisect --auto-sqlite-run results.sqlite --parallel=1 --jobs=1
Before parallel bisect run, compile all object files
  (1 of 5) clang++ -O3 -freciprocal-math:  done
  (2 of 5) clang++ -O3 -funsafe-math-optimizations:  done
  (3 of 5) g++ -O3 -freciprocal-math:  done
  (4 of 5) g++ -O3 -funsafe-math-optimizations:  done
  (5 of 5) g++ -O3 -mavx2 -mfma:  done
Updating ground-truth results - ground-truth.csv - done

Run 1 of 5
flit bisect --precision double "clang++ -O3 -freciprocal-math" LuleshTest
Updating ground-truth results - ground-truth.csv - done
Searching for differing source files:
[...]
```

(takes approximately 3 min 10 sec)

Will automatically run all rows with `comparison > 0.0`

Let's look at the Bisect algorithm

How to Perform the Search

- **Problem:** search space is exponential
- **Problem:** floating-point errors combine in non-intuitive ways

Assumption 1: errors do not exactly cancel

- **Delta Debugging:** old but good idea $O(n \log n)$

Assumption 2: variability sites act alone

- **Linear Search:** simple $O(n)$
- **Logarithmic Search:** find one at a time $O(k \log n)$

Bisect Algorithm

- Simple divide and conquer
- Guaranteed to have no false positives
- False negatives identified automatically

Algorithm 1 Bisect Algorithm

```
procedure BisectAll(Test, items)
    found ← { }
    T ← Copy(items)
    while Test(T) > 0 do
        G, next ← BisectOne(Test, T)
        found ← found ∪ next
        T ← T \ G
    assert Test(items) = Test(found)
    return found
```

```
procedure BisectOne(Test, items)
    if Size(items) = 1 then                          ▷ base case
        assert Test(items) > 0
        return items, items
    Δ1, Δ2 ← SplitInHalf(items)
    if Test(Δ1) > 0 then
        return BisectOne(Test, Δ1)
    else
        G, next ← BisectOne(Test, Δ2)
        return G ∪ Δ1, next
```

Exercise 3 - `./step-14.sh`

```
exercise-3 $ head -n 3 auto-bisect.csv
testid,bisectnum,compiler,optl,switches,precision,testcase,type,name,return
1,1,clang++,-O3,-freciprocal-math,double,LuleshTest,completed,"lib,src,sym",0
1,1,clang++,-O3,-freciprocal-math,double,LuleshTest,src,"('tests/LuleshTest.cpp',
0.33294020544031533)",0
```

Results are placed in a CSV file for easy access

Exercise 3 - Bonus

Exercise 3 - efficiency

The 4th run (from auto-run) took 34 compilation / run steps.

```
Run 4 of 5
flit bisect --precision double "g++ -O3 -funsafe-math-optimizations" LuleshTest
  [...]
All variability inducing symbols:
  ../packages/LULESH/lulesh-init.cc:16 _ZN6DomainC1Eiiiiiiiii -- Domain::Domain(int, int, int,
int, int, int, int, int, int) (score 2.3302358973548727)
  ../packages/LULESH/lulesh-init.cc:219 _ZN6Domain9BuildMeshEiii -- Domain::BuildMesh(int, int,
int) (score 1.4315005606175104)
  ../packages/LULESH/lulesh.cc:1362 _Z14CalcElemVolumePKdS0_S0_ -- CalcElemVolume(double const*,
double const*, double const*) (score 0.9536115035892543)
  ../packages/LULESH/lulesh.cc:1507 _Z22CalcKinematicsForElemsR6Domaindi --
CalcKinematicsForElems(Domain&, double, int) (score 0.665781828022106)
  ../packages/LULESH/lulesh.cc:2651 _Z11lulesh_mainiPPc -- lulesh_main(int, char**) (score
0.3328909140110529)
```

Can we do better?

What if we only want the **top contributing function**?

Exercise 3 - `./step-15.sh`

```
exercise-3 $ flit bisect --biggest=1 --precision=double "g++-7 -O3 -funsafe-math-optimizations"
LuleshTest
Updating ground-truth results - ground-truth.csv - done
Looking for the top 1 different symbol(s) by starting with files
  [...]
    Found differing source file ../packages/LULESH/lulesh-init.cc: score 3.7609285311270604
    Searching for differing symbols in: ../packages/LULESH/lulesh-init.cc
      [...]
        Found differing symbol on line 16 -- Domain::Domain(int, int, int, int, int, int, int,
int, int) (score 2.3302358973548727)
  [...]
  Created ./bisect-06/bisect-make-20.mk - compiling and running - score 0.022750390077923448
    Found differing source file tests/LuleshTest.cpp: score 0.022750390077923448
[...]
The 1 highest variability symbol:
  ../packages/LULESH/lulesh-init.cc:16 _ZN6DomainC1Eiiiiiiiii --  Domain::Domain(int, int, int,
int, int, int, int, int, int) (score 2.3302358973548727)
```

- Found the same highest variability function: `Domain::Domain()`
- Found it in 20 compile/run cycles instead of 34
- Searches for symbols after each file

Thank You!

Questions?

pruners.github.io/flit