FLiT
Measuring and Locating Floating-Point Variability from Compiler Optimizations

Ignacio Laguna, Harshitha Menon
Lawrence Livermore National Laboratory

Michael Bentley, Ian Briggs, Pavel Panchekha, Ganesh Gopalakrishnan
University of Utah

Hui Guo, Cindy Rubio González
University of California at Davis

Michael O. Lam
James Madison University

http://fpanalysistools.org/
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

http://fpanalysistools.org/
Example of Compiler-Induced Variability

Laghos: A high-order Lagrangian hydrodynamics mini-application

\texttt{xlc -O2} \quad \texttt{xlc -O3}

One iteration: \textbf{11.2}\% relative error!
And speedup by a factor of \textbf{2.42}

What happened? How can I investigate it?

http://fpanalysistools.org/
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

```bash
$ git clone https://github.com/PRUNERS/FLiT.git
Cloning into 'FLiT'...  
[...]
$ 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
[...]
```
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

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

<table>
<thead>
<tr>
<th>Source files</th>
<th>97</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average functions per file</td>
<td>31</td>
</tr>
<tr>
<td>Total functions</td>
<td>2,998</td>
</tr>
<tr>
<td>Source lines of code</td>
<td>103,205</td>
</tr>
</tbody>
</table>
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

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

tests/Mfem13.cpp

```
Module-FLiT $ cd exercise-1

exercise-1 $ vim tests/MFEM13.cpp

or

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

or whatever...
Exercise 1 - Create MFEM Test

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

```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);
```
Exercise 1 - Create MFEM Test

- A simple test setup with no floating-point inputs
- `compare()` does L2 norm and returns % relative difference (skipped)
Exercise 1 - Create MFEM Test

- Only double precision is implemented
- Create a temporary directory and go there (for out files)
Exercise 1 - Create MFEM Test

Tests/MFEM13.cpp

```cpp
74  // Run the example's main under MPI
75  auto meshfile = flit::join(start_dir, "data", "beam-tet.mesh");
76  auto result = flit::call_mpi_main(
77      mfem_13p_main,
78      "mpirun -n 1 --bind-to none",
79      "Mfem13",
80      "--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

- 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

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

```cpp
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);
```

http://fpanalysistools.org/
Exercise 1 - Create MFEM Test

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

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

```
flit-config.toml

1 [run]
2 enable_mpi = true
```
Exercise 1 - FLiT Configuration

```toml
flit-config.toml

4 [dev_build]
5  compiler_name = 'g++'
6  optimization_level = '-O3'
7  switches = '-mavx2 -mfma'

8
9 [ground_truth]
10 compiler_name = 'g++'
11 optimization_level = '-O2'
12 switches = ''
```

Defines the compilations for `make dev` and `make gt`
Exercise 1 - FLiT Configuration

- Defines the “g++” compiler
- Defines the compilation search space
Exercise 1 - FLiT Configuration

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

---

```toml
[[compiler]]
binary = 'clang++-6.0'
name = 'clang++'
type = 'clang'
optimization_levels = ['-O3',

switches_list = ['-ffast-math',
'-funsafe-math-optimizations',
'-mfma',

flit-config.toml
```
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)

```bash
exercise-1 $ vim custom.mk
```
Exercise 1 - Makefile Configuration

custom.mk

4 PACKAGES_DIR := $(abspath ../packages)
5 MFEM_SRC := $(PACKAGES_DIR)/mfem
6 HYPRE_SRC := $(PACKAGES_DIR)/hypre
7 METIS_SRC := $(PACKAGES_DIR)/metis-4.0
8
9 SOURCE :=
10 SOURCE += $(wildcard *.cpp)
11 SOURCE += $(wildcard tests/*.cpp)
12
13 # Compiling all sources of MFEM into the tests takes too long for a tutorial
14 # skip it. Instead, we link in the MFEM static library
15 #SOURCE += $(wildcard ${MFEM_SRC}/fem/*.cpp)
16 #SOURCE += $(wildcard ${MFEM_SRC}/general/*.cpp)
17 #SOURCE += $(wildcard ${MFEM_SRC}/linalg/*.cpp)
18 #SOURCE += $(wildcard ${MFEM_SRC}/mesh/*.cpp)
19
20 # just the one source file to see there is a difference
21 SOURCE += ${MFEM_SRC}/linalg/densemat.cpp  # where the bug is
Exercise 1 - Makefile Configuration

custom.mk

23 CC_REQUIRED += -I${MFEM_SRC}
24 CC_REQUIRED += -I${MFEM_SRC}/examples
25 CC_REQUIRED += -isystem ${HYPRE_SRC}/src/hypre/include
26
27 LD_REQUIRED += -L${MFEM_SRC} -lmfem
28 LD_REQUIRED += -L${HYPRE_SRC}/src/hypre/lib -lHYPRE
29 LD_REQUIRED += -L${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

http://fpanalysisstools.org/
Exercise 1 - ./step-01.sh

- 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[...]
 [...]
```

(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_FFA[
results/GCC_ip-172-31-8-101_FUNSAFE_MATH_OPTIMIZATIONS_O3-out -> results/GCC_ip-
results/GCC_ip-172-31-8-101_MFMA_O3-out -> results/GCC_ip-172-31-8-101_MFMA_o[
results/CLANG_ip-172-31-8-101_FFAST_MATH_O3-out -> results/CLANG_ip-172-31-8-101[
results/CLANG_ip-172-31-8-101_FUNSAFE_MATH_OPTIMIZATIONS_O3-out -> results/CLANG[
results/CLANG_ip-172-31-8-101_MFMA_O3-out -> results/CLANG_ip-172-31-8-101_MFMA_[
[...]
```

(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

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

<table>
<thead>
<tr>
<th>compiler</th>
<th>optl</th>
<th>switches</th>
<th>comparison</th>
<th>nanosec</th>
</tr>
</thead>
<tbody>
<tr>
<td>clang++-6.0</td>
<td>-03</td>
<td>-ffast-math</td>
<td>0.0</td>
<td>2857386994</td>
</tr>
<tr>
<td>clang++-6.0</td>
<td>-03</td>
<td>-funsafe-ma</td>
<td>0.0</td>
<td>2853588952</td>
</tr>
<tr>
<td>clang++-6.0</td>
<td>-03</td>
<td>-mfma</td>
<td>0.0</td>
<td>2858789982</td>
</tr>
<tr>
<td>g++-7</td>
<td>-03</td>
<td>-ffast-math</td>
<td>0.0</td>
<td>2841191528</td>
</tr>
<tr>
<td>g++-7</td>
<td>-03</td>
<td>-funsafe-ma</td>
<td>0.0</td>
<td>2868636192</td>
</tr>
<tr>
<td>g++-7</td>
<td>-03</td>
<td>-mfma</td>
<td>193.007351</td>
<td>2797305220</td>
</tr>
</tbody>
</table>

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

Compilation: g++-7 -03 -mfma
What’s Different?

```bash
exercise-2 $ diff -u ../exercise-1/custom.mk ./custom.mk
+++ 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

```sh
equation-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

[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

- baseline (e.g., g++ -O0)
- under test (e.g., g++ -O3)
- final executable (mixed)
Exercise 2 - Bisect Details

Approach: combine symbols after compilation

Convert function symbols into weak symbols

baseline (e.g., g++ -O0)

under test (e.g., g++ -O3)

final executable (mixed)

Downside: Requires recompiling with -fPIC
Exercise 2 - ./step-10.sh

eexercise-2 $ cat -n ./packages/mfem/linalg/densemat.cpp | tail -n +3688 | head -n 24
3688 void AddMult_a_AAt(double a, const DenseMatrix &A, DenseMatrix &AAt)
3689 {
3690     double d;
3691     for (int i = 0; i < A.Height(); i++)
3692         {
3693             for (int j = 0; j < i; j++)
3694                 {
3695                     d = 0.;
3696                     for (int k = 0; k < A.Width(); k++)
3697                         {
3698                             d += A(i,k) * A(j,k);
3699                         }
3700                         AAt(i, j) += (d *= a);
3701                         AAt(j, i) += d;
3702                     }
3703                 }
3704                 d = 0.;
3705                 for (int k = 0; k < A.Width(); k++)
3706                     {
3707                         d += A(i,k) * A(i,k);
3708                     }
3709                 AAt(i, i) += a * d;
3710             }
3711         }

Computes

\[ M = M + aAA^T \]
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

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;

<table>
<thead>
<tr>
<th>compiler</th>
<th>optl</th>
<th>switches</th>
<th>comparison</th>
<th>nanosec</th>
</tr>
</thead>
<tbody>
<tr>
<td>clang++-6.0</td>
<td>-03</td>
<td>-freciprocal-math</td>
<td>5.52511478433538e-05</td>
<td>432218541</td>
</tr>
<tr>
<td>clang++-6.0</td>
<td>-03</td>
<td>-funsafe-math-opt</td>
<td>5.52511478433538e-05</td>
<td>432185456</td>
</tr>
<tr>
<td>clang++-6.0</td>
<td>-03</td>
<td></td>
<td>0.0</td>
<td>433397072</td>
</tr>
<tr>
<td>g++-7</td>
<td>-03</td>
<td>-freciprocal-math</td>
<td>5.52511478433538e-05</td>
<td>441362811</td>
</tr>
<tr>
<td>g++-7</td>
<td>-03</td>
<td>-funsafe-math-opt</td>
<td>7.02432004920159</td>
<td>436202864</td>
</tr>
<tr>
<td>g++-7</td>
<td>-03</td>
<td>-mavx2 -mfma</td>
<td>1.02330009691563</td>
<td>416599918</td>
</tr>
<tr>
<td>g++-7</td>
<td>-03</td>
<td></td>
<td>0.0</td>
<td>432654778</td>
</tr>
</tbody>
</table>

sqlite> .q

Five variability compilations.
Let’s investigate!
Exercise 3 - ./step-12.sh

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

```plaintext
1: procedure BisectAll(Test, items)
2:    found ← { }
3:    T ← COPY(items)
4:    while Test(T) > 0 do
5:        G, next ← BisectOne(Test, T)
6:        found ← found ∪ next
7:        T ← T \ G
8:        assert Test(items) = Test(found)
9:    return found
```

```plaintext
1: procedure BisectOne(Test, items)
2:    if SIZE(items) = 1 then  ▶ base case
3:        assert Test(items) > 0
4:        return items, items
5:    Δ₁, Δ₂ ← SplitInHalf(items)
6:    if Test(Δ₁) > 0 then
7:        return BisectOne(Test, Δ₁)
8:    else
9:        G, next ← BisectOne(Test, Δ₂)
10:    return G ∪ Δ₁, next
```

http://fpanalysistools.org/
Exercise 3 - ./step-14.sh

Results are placed in a CSV file for easy access
Exercise 3 - Bonus
Run 4 of 5
flit bisect --precision double "g++ -O3 -funsafe-math-optimizations" LuleshTest
[...]
All variability inducing symbols:
../packages/LULESH/lulesh-init.cc:16 _ZN6DomainC1Eiiiiiiiiii -- Domain::Domain(int, int, int, int, int, int, int, int, int) (score 2.3302358973548727)
../packages/LULESH/lulesh-init.cc:219 _ZN6Domain9BuildMeshEiiii -- 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 _Z22CalcKinematicsForElemsR6Domainidi -- 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 _ZN6DomainC1Eiiiiiiiiii -- 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