ECE 6930-004 HPC Fault Tolerance

WHAT IS A SOFT ERROR?

DR. JON C. CALHOUN

Paper selection Unit 3: Soft errors

Date	Paper/Topic	Presenter
9/27	What is a soft error? / Fault Injection Techniques	Calhoun
10/2	Coding Techniques	Calhoun
10/4	Memory Errors in Modern Systems: The Good, The Bad, The Ugly	
10/9	Algorithm-Based Fault Tolerance for Matrix Operations	
10/11	Shoestring: Probabilistic Soft Error Reliability on the Cheap	
10/16	Detection and Correction of Silent Data Corruption for Large-Scale High-Performance Computing	
10/18	Energy Analysis and Optimization for Resilient Scalable Linear Systems	Miao

Schadenfreude!

Situation: You preform floating-point division inside your code

Task: Do you trust the results?

Pentium FDIV bug:

- Bug in FDIV instruction on early Pentium processors in 1994 caused incorrect results
- Discovered by Prof. Thomas Nicely at Lynchburg College
- Attributed to missing entries in the lookup table used by the floating-point division circuitry
- Intel claimed common users would expose bug once very 27,000 years
- Byte magazine estimated 1 in 9 billion divides

$$\frac{4{,}195{,}835}{3{,}145{,}727} = 1.333820449136241002$$

$$\frac{4{,}195{,}835}{3{,}145{,}727} = 1.333739068902037589$$

Outline

Review of causes of errors

Where do they come from?

Cosmic Rays

Outcomes

Summary

Review: Error characteristics

Errors indicate the presence of incorrect state in the system

Faults -> Errors -> Failures

Errors are classified as

- **Detected:** indicated by error message or signal
- **Latent/silent:** not detected
- **Masked:** not causing a failure
- **Soft:** due to a transient fault

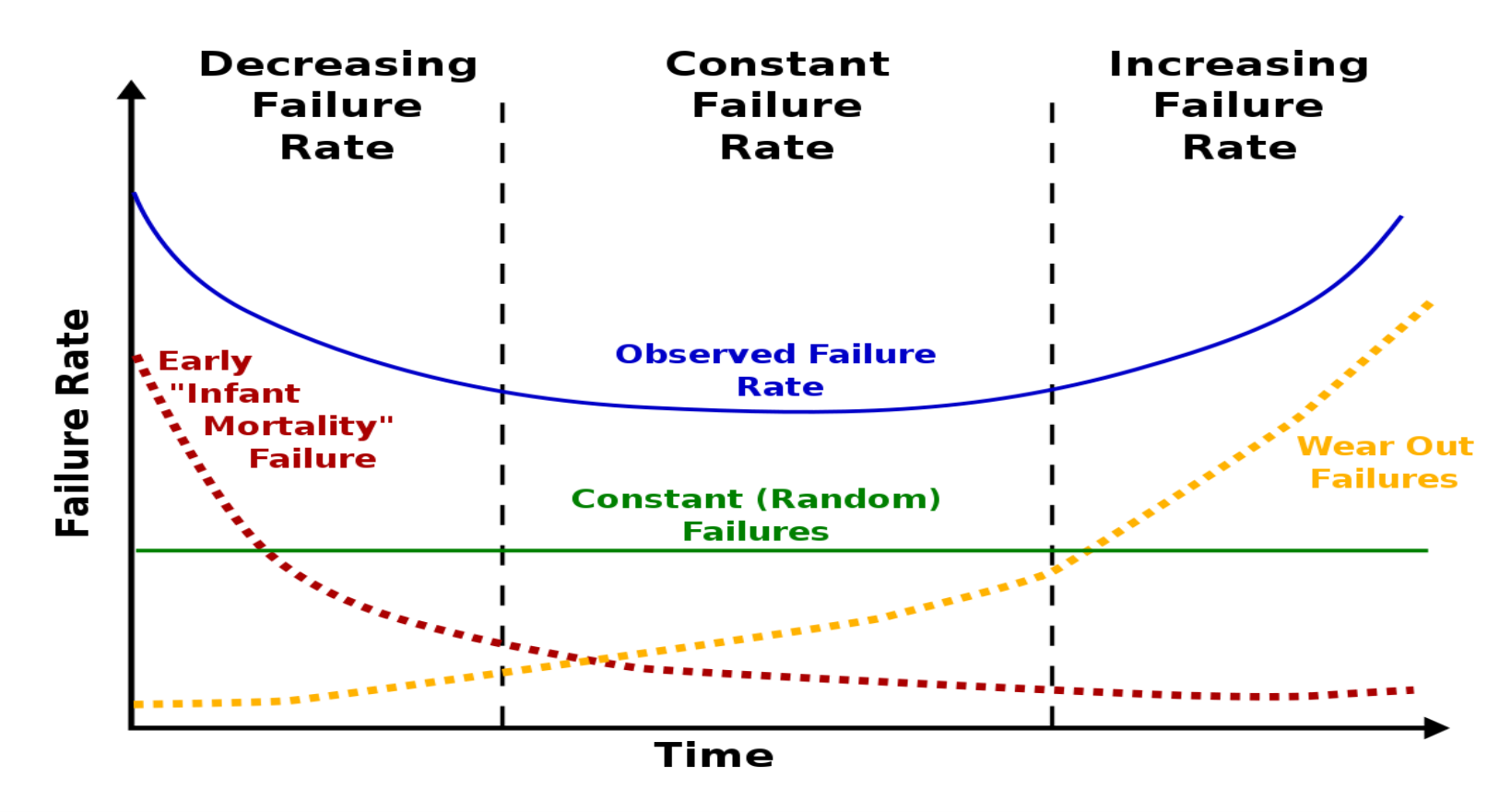
What is an error?

What causes errors?

How do we classify errors?

Breaking down a system's lifetime

"Bathtub Curve" of observed failure rate [Wikipedia 2017]



Where do errors come from?

HPC systems are extremely complex and can come from any component or sub-component. At a low level errors can be mapped to:

- Dielectric breakdown and electrical breakdown
- Temperature (extremes and variations)
- Aging
- Manufacturing defects
  Stress

- Extreme conditions
- Voltage fluctuation
- Electro-magnetic interference
- Terrestrial neutrons
- Cosmic radiation
- Alpha particles

Primary Focus

Cosmic Rays

Muons (very heavy electrons)

- Most abundant particle in shower
- Deposits energy evenly during collisions
- Don't do much damage to electrical circuits

Neutrons

- ~70/hour per centimeter at Los Alamos National Laboratory
- Most of the time it goes right through matter
- When they hit nuclei they emit nuclear fragments
  - Pions, protons, alpha particles, other heavy nuclei

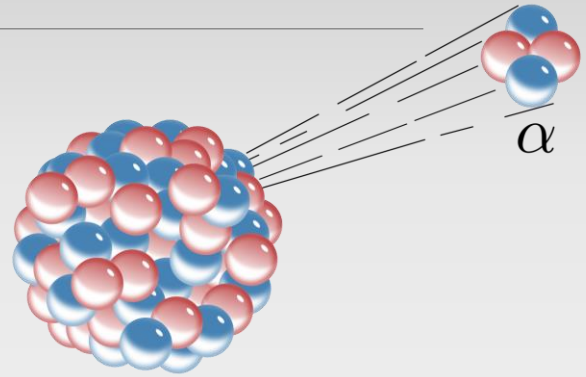
Radiation

- Your body protects repairs itself
- Computes don't have that ability

More Radiation

Alpha particles are another source of soft errors

- Alpha particles have the same make up as a Helium atom
- Emitted from radioactive decay

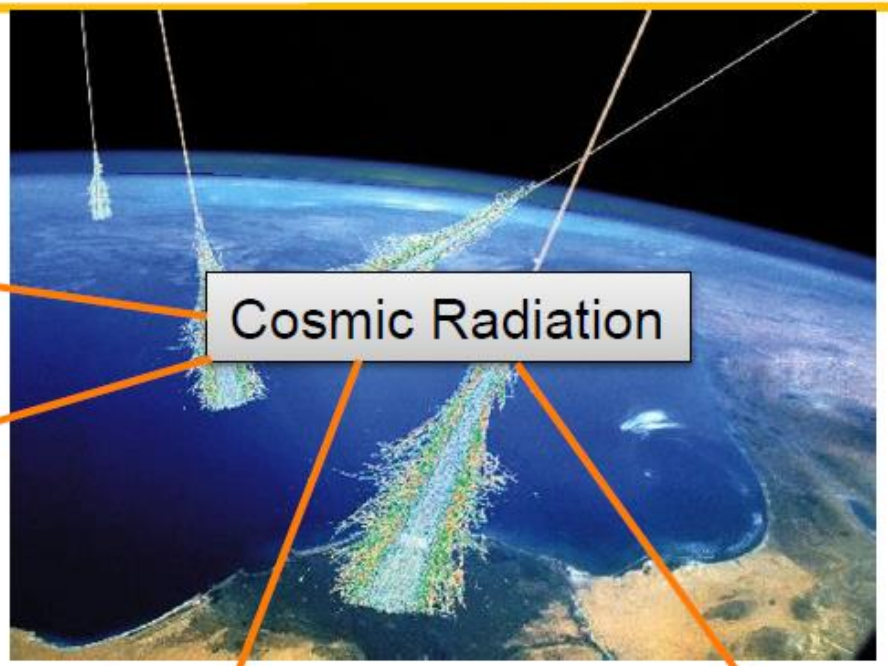


Two main sources:

- **Manufacturing**
  - 1978 Intel failed to deliver chips to AT&T due to trace amounts of uranium in the source materials
  - 1986 IBM used radioactive contaminate to clean bottles used to store an acid needed in chip manufacturing

- **Cosmic Radiation:**
  - Charged alpha particles interact directly with electrons forming “electron-hole pairs”

Computer Effects: Cosmic Radiation Example



TRANSIENT

Bit Flips

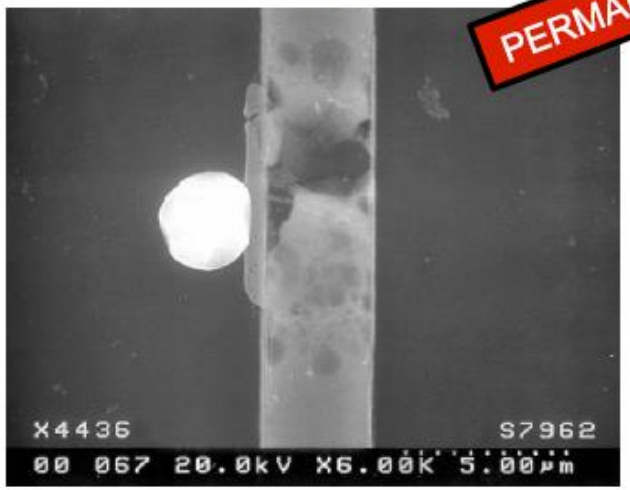
1011101 → 100110

Single bit flips are corrected by ECC, double bit flips are detected, more are uncaught

Error Correcting Codes – hardware implemented in memories

Stuck Bits

PERMANENT



Damage caused by single event latchup

TRANSIENT

Burst Errors

Many bit flips on an entire page of memory. **Not caught by ECC.** Attributed to memory logic (refresh / control)

Stuck Pages

Largely an unknown

PERMANENT

Source: LANL

Outcome of cosmic radiation

Crashes

- Hardware/Software crashes and much be restarted/reset
  - Use checkpoint-restart!

Performance variation

- Wastes time/energy
- What is the difference between this and system noise?

Data corruption

- Wrong results!
- What if the results are only slightly off?
  - 2.0 vs 2.000001

If data corruption is "silent", how do you know it happened?

How often do these occur?

These events occur on a daily basis

More common in memories than logic

- Memories occupy most on chip area

Transistor susceptibility impacted by feature size and supply voltage

- Smaller area means, when they hit they hit hard
- Lower voltage leads to timing issues or higher probability of bit-flips

We'll explore how much in the next paper

We will explore coding techniques for memories next class

Summary

Soft errors that silently corrupt data are a concern on future HPC system

- No one wants to waste a few days of compute time and get wrong answers

Reliance on off-the-shelf hardware not designed for the extreme-scales is increasing the rate of soft errors

- We need some way to protect applications

Our next group of paper will focus first on the frequency of soft errors and then on software techniques to guard against them

ECE 6930-004 HPC Fault Tolerance

OVERVIEW OF FAULT INJECTION TECHNIQUES FOR HPC

DR. JON C. CALHOUN

Overview

Fail-stop failures

- Ways to inject

Transient faults

- Ways to inject
- Overview of FlipIt
- Overview of FaultSight

Fail-stop

Application or some system resource becomes unresponsive and no longer provides its intended service

Type of failure that most people consider when they think of a failure in HPC systems

- Readily detectable and correctable
- Best understood type of failures

HEARTBEATS

How do we detect fail-stop failures?

Fail-stop

Simulating a process halt is often sufficient for this class of failures

Direct access to job:

- `kill -9`
- Debugger to send signal to process, or corrupt data leading to a crash

Modifying the program:

- Program manually raises signal
- Manually generate segfault or assert violation in code

Fail-stop

Wrappers allow for abstraction of underlying HPC components:

PMPI
- Down node
- Network outage

HDF5
- File system unreachable
- Reduced performance

LD_PRELOAD
- Failure of system level componets

Fail-stop

What are some other ways that you can emulate or induce fail-stop failures?

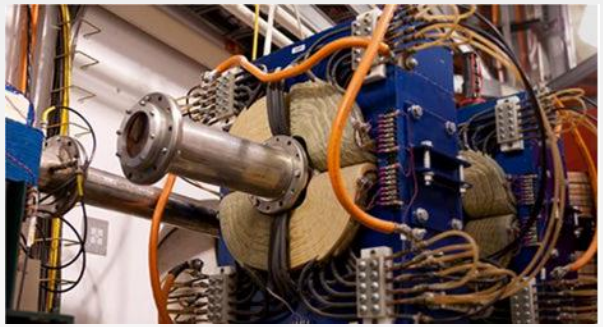
Transient – low level

The most accurate way to simulate silent errors is with a neutron beam:

+ Accurate
- High cost

Cycle accurate architectural simulators offer more control over on error location (e.g. SIMICS)

+ Deep control
+ Much lower cost and overhead
- Still much slower than actual code
- Most injected faults are masked before affecting application



Radiation Beam [Carsile 2015]

Transient – medium level

Running code inside a virtual machine allows visibility of all executed instructions and guest physical addresses-e.g. F-SEFI [Guan et al. 2014]

+ Full control of application, OS, and all software running on the node
- May need to bridge semantic gap to know where injections should occur
- Large overhead

Debugger's ability to inspect and modify system state can be useful for silent error injection (e.g. GDB, TotalView, DDT)

+ Excellent control on injection location within application source code
+ Scalable to large core counts
- Needs explicit injection points and conditions

Transient – medium level

Modify the binary at runtime using binary instrumentation tools such as Intel Pin

- \+ Good control over how injection occurs
- \- Architecture specific

Compile based approaches modify code generated by a compiler to inject faults statically or dynamically e.g., KULFI [Sharma et al. 2013], FlipIt [Calhoun et al. 2014], LLFI [Lu et al. 2015]

- \+ Stuck at faults and transient errors
- \+ Architecture agnostic
- \- Increased runtime compared to normal application
- \- Injections may not be one-to-one with executed instructions

Transient – high level

Most HPC applications checkpoint to recover from failures

Modifying a checkpoint file from inside the checkpoint/restart routine or with a script allows for corruption in state variables [Benson et al. 2014] [Gomez et al 2014, 2015, 2016]

- + Low overhead
- + Change can be understood mathematically
- - May not accurately reflect how errors propagate from hardware to software

PMPI can be used to simulate errors in the network by modifying data before it's set/received [Lu and Reed 2004][Fiala et al 2012]

FlipIt

LLVM based fault injector that instruments code as it is being compiled

Faults are injected at runtime based on user's custom runtime parameters

Allows only a subset of code and MPI processes to be faulty

Simulates processor logic and memory errors

FlipIt is currently maintained by:

- Jon Calhoun
- Luke Olson

https://github.com/joncalhoun40/FlipIt

FlipIt - Compile Time Under the Hood

```
for all instructions in Function
        if should instrument instruction
                instrument();
                logInfo();
```

What does the compiler pass do to my code?

FlipIt - Compile Time Under the Hood (Instrument)

%C = **fmul** double %B, %A;

%result = **fadd** double %C, %D;

FlipIt - Compile Time Under the Hood (Instrument)

```
%C = fmul double %B, %A;
%crptd = call double @crptDbl(double %C, ...);
%result = fadd double %C, %D;
```

FlipIt - Compile Time Under the Hood (Instrument)

```
%C = fmul double %B, %A;
%crptd = call double @crptDbl(double %C, ...);
%result = fadd double %crptd, %D;
```

FlipIt - Compile Time Under the Hood

A log file is generated for each file compiled with FlipIt:

- Relate fault injection sites back to source code
- Provide extra information for visualization

```
Function Name: waxpby
 ------------------------------------------------------------------------
 #1291   FCmp   Arg 0   Control-Branch  /home/jcalhoun/HPCCG-1.0/waxpby.cpp:53
 #1292   ICmp   Arg 1   Control-Branch  /home/jcalhoun/HPCCG-1.0/waxpby.cpp:57
 #1293   Add    Result  Arith-Fix       /home/jcalhoun/HPCCG-1.0/waxpby.cpp:57
 #1294   Add    Result  Arith-Fix       /home/jcalhoun/HPCCG-1.0/waxpby.cpp:57
 #1295   Add    Result  Arith-Fix       /home/jcalhoun/HPCCG-1.0/waxpby.cpp:57
 #1296   And    Result  Arith-Fix       /home/jcalhoun/HPCCG-1.0/waxpby.cpp:57
 #1297   ICmp   Arg 0   Control-Branch  /home/jcalhoun/HPCCG-1.0/waxpby.cpp:57
 ...
```

FlipIt - Runtime Under the Hood

The corrupt* functions perform the heavy lifting inside FlipIt

```
corrupt(info, prob, data)
        if shouldInject()
                return data xor 0x1 << x;
        return data;
```

```
shouldInject()
    if injector active
        && rank faulty
        && fault site active
        && prob > calcProb()
        return true;
    return false;
```

FlipIt - Runtime Under the Hood

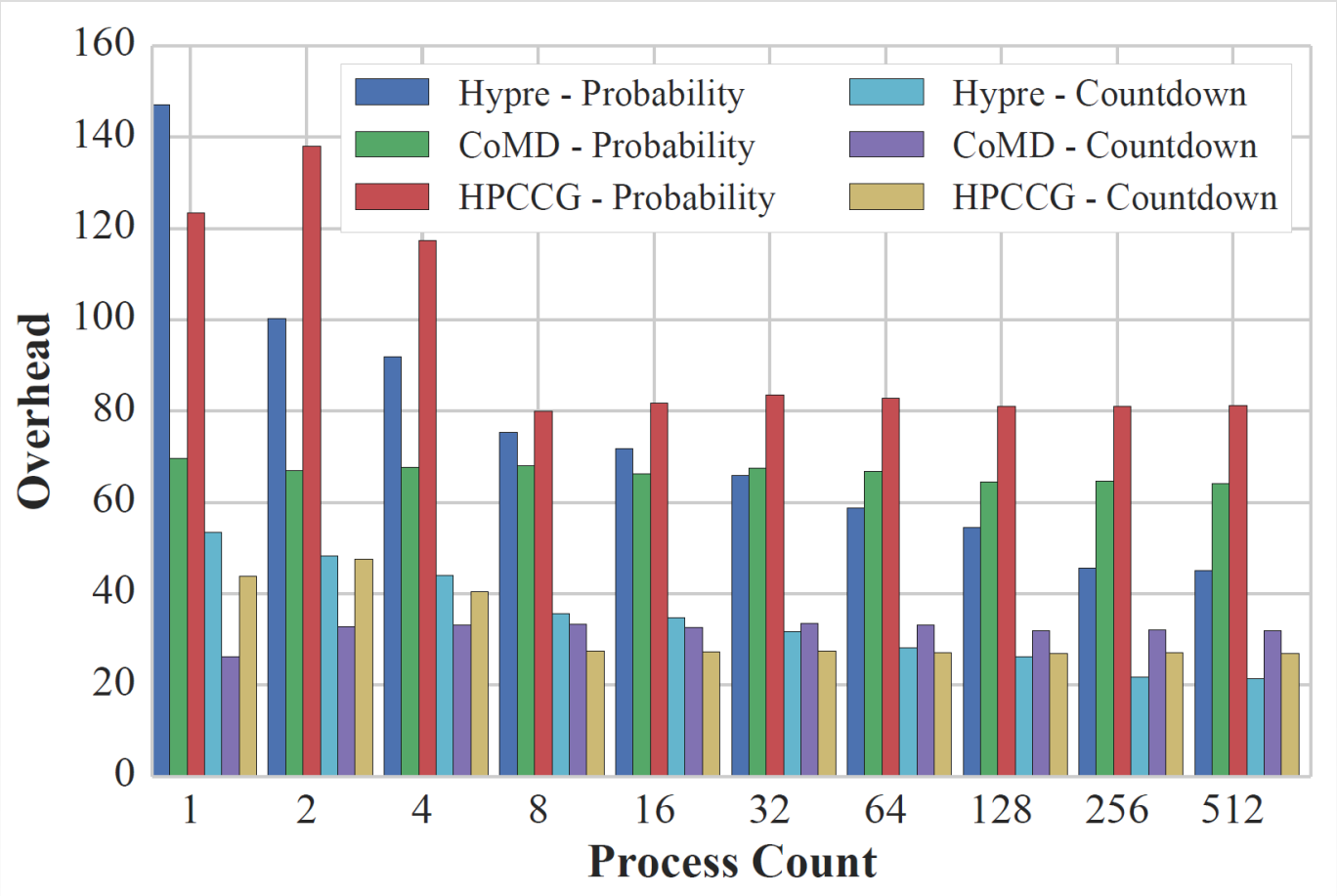
FlipIt is design to be extremely customizable supporting:

- Which MPI rank(s) are faulty
- Varying probabilities of injection for every instruction type
- Injection in only certain types of instructions
- Custom probability distributions
- Custom logging functions on injection
- Which bit and byte injection should occur in
- Number of Injections

FlipIt - Performance

Overhead lowers as more processors used

Countdown timer less overhead than probability calculation



FlipIt - Usability

Designed to require minimal modifications to application < 10 LOC

HPCCG Modifications:

HPCCG.cpp

```
#include "FlipIt/corrupt/corrupt.h"
…
int main(int argc, char* argv[]){
…
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
FLIPIT_Init(rank, argc, argv, time(NULL));
…
FLIPIT_Finalize(NULL);
MPI_Finalize();
```

Makefile

```
CXX = $(FLIPIT_PATH)/scripts/flipit-cc
LINKER = $(FLIPIT_PATH)/scripts/flipit-cc
CPP_OPT_FLAGS = -O2 -g
```

FlipIt/scripts/config.py

```
# set mpicc includes paths and libraries for machine
SHOW = " -I/usr/local/include -L/usr/local/lib”
       “ -lmpich -lopa -lmpl -lrt -lpthread “
CPP_LIB = " -ldl -L/usr/lib/ -lm -lstdc++ “
```

FlipIt - `flipit-cc`

Simplifies the compilation process by wrapping the following calls

1. `clang -emit-llvm -c` (Compile to bit-code)
2. `llvm-link` (Link with external bit-code)
3. `opt -FlipIt` (Instrument bit-code)
4. `clang -o` (Compile to object code)

Parameters for library locations and parameters for the compiler pass are set in a config file

FlipIt - `config.py`

Provides configurations used during compile time

opt :

- `opt` FlipIt config file
- Function names to instrument
- Probability of a fault
- Faulty byte number
- Faulty bit number
- Pointer injections
- Arithmetic injections
- Control injection
- State file location

clang :

- Compler command wrapped by `mpicc`
- C++ library location
- Files not to instrument
- Default compiler

FlipIt - Runtime API

```
void FLIPIT_SetInjector(int state);
void FLIPIT_SetRankInject(int state);

void FLIPIT_SetFaultProbability(double(faultProb)());
void FLIPIT_SetCustomLogger(void (customLogger)(FILE*));
void FLIPIT_CountdownTimer(unsigned long numInstructions);

unsigned long long FLIPIT_GetExecutedInstructionCount();
int FLIPIT_GetInjectionCount();
void FLIPIT_SetMaxInjections(int n);
int FLIPIT_GetMaxInjections();
```

HW3

You will use FlipIt to inject bit-flips into an HPC application on a supercomputer

More to come!!!

FaultSight

Fast efficient way to analyze fault injection campaigns conducted by fault injection frameworks
- Here we will use it with FlipIt

Python based web app developed and maintained by:
Jon Calhoun
Einar Horn

https://github.com/einarhorn/FaultSight

Let us know what you think as we are actively developing this tool!

FaultSight - Overview

1. Reads the compile logs generated by FlipIt to determine fault sites, their properties, and their relationship with the original source code

2. Extracts runtime time injection information from each trial in the fault injection campaign logged in the each trial's output file

3. Combines all this into a database that is dynamically queried to generate graphs

FaultSight - What do we store in the database?

Site Info:

- Site number
- Type of injection
- Opcode
- File, function, line number

Trial Info:

- Trial number
- Crashed
- Detected
- Signals

Injection Info:

- Site and trial number
- MPI rank
- Bit flipped
- Cycle

Application specific information:

- Iteration
- Converged
- Etc.

FaultSight-API

Provides wrappers around insertions into the database

1. Create and connect to database
2. Insert information about possible fault injection locations
3. Record events of each fault injection trial
   - Injections
   - Signals
   - Detections

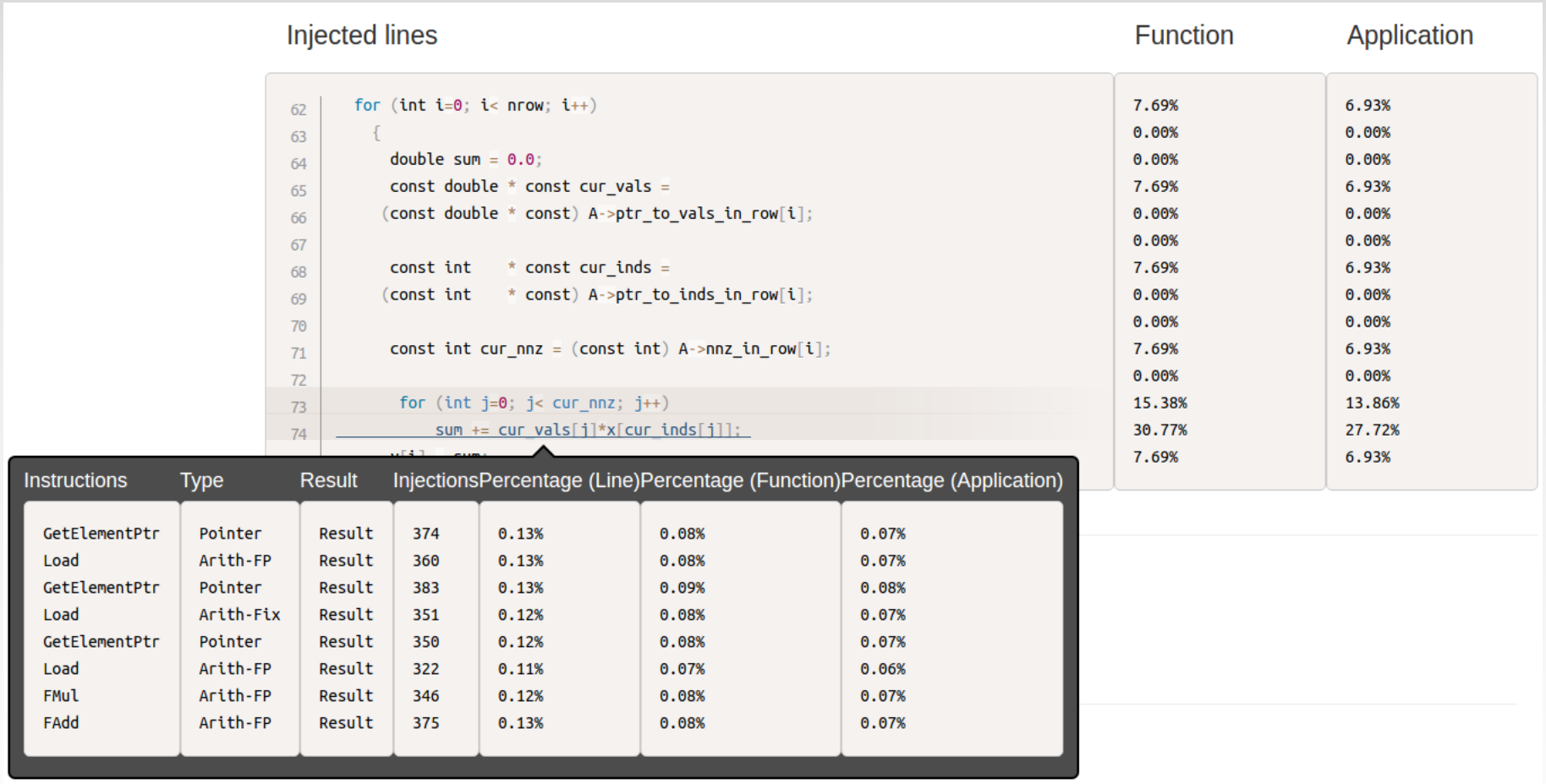
FaultSight - Analysis

Plethora of graphs:

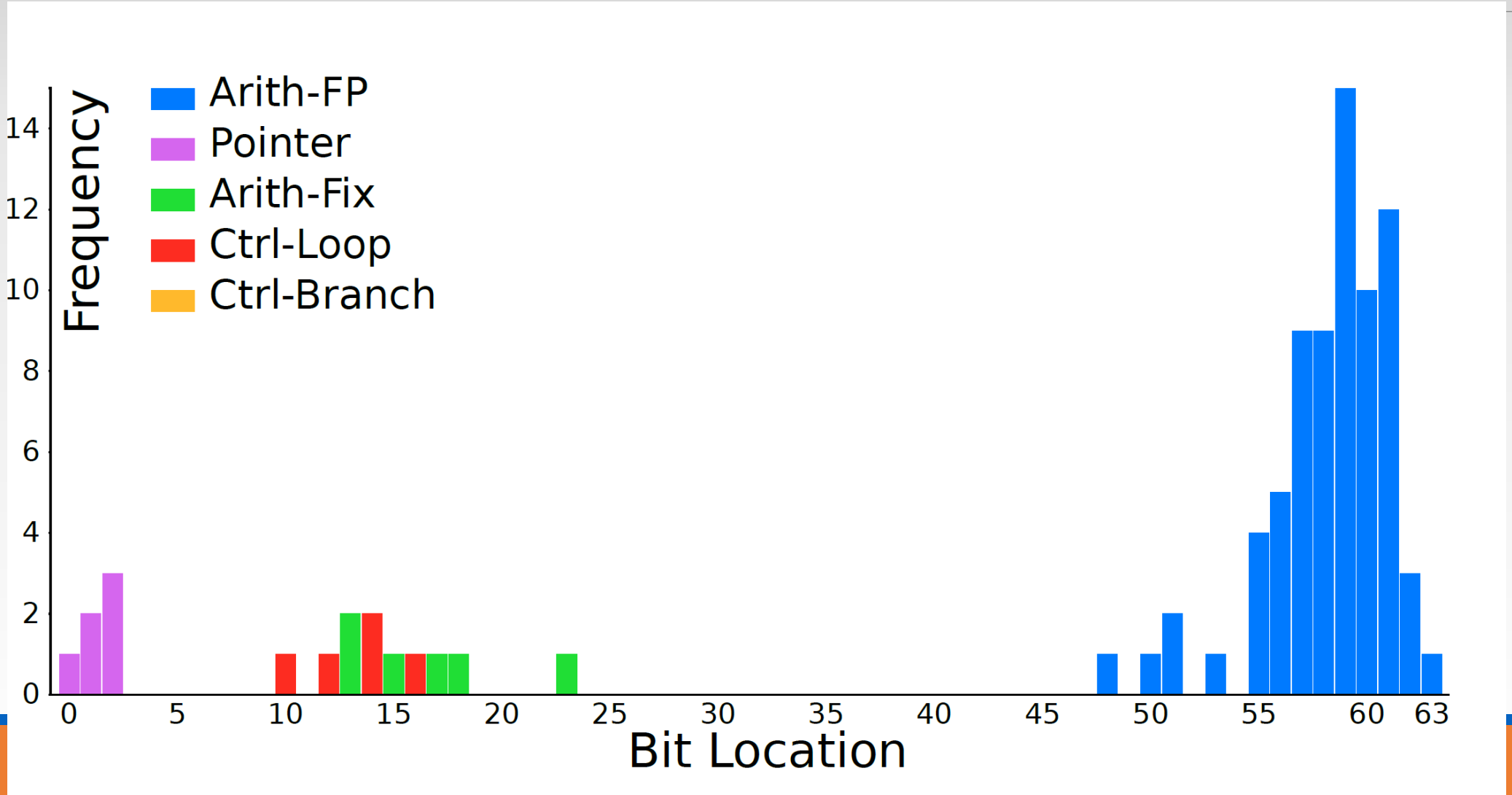
- Classification of injections based on instruction type
- Percent of injections based on function (includes type information
- Signals generated
- Trials with detection (includes bit locations and types)
- Latency of detection
- Trials that unexpectedly terminate (includes bit locations and types)

User custom graphs that can view the above with per function, per line, per time-step resolution

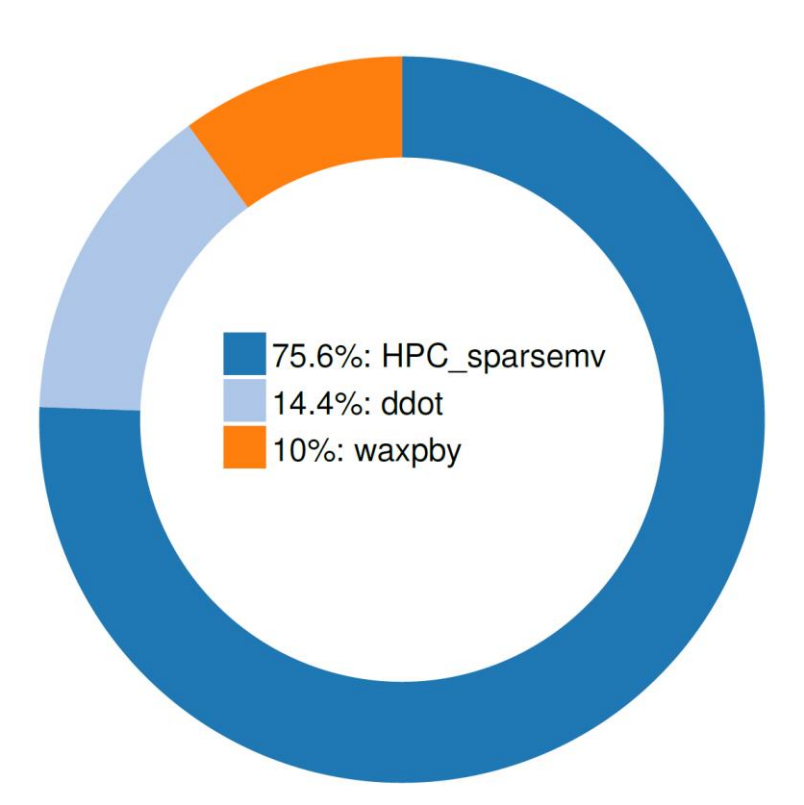
Code profiler



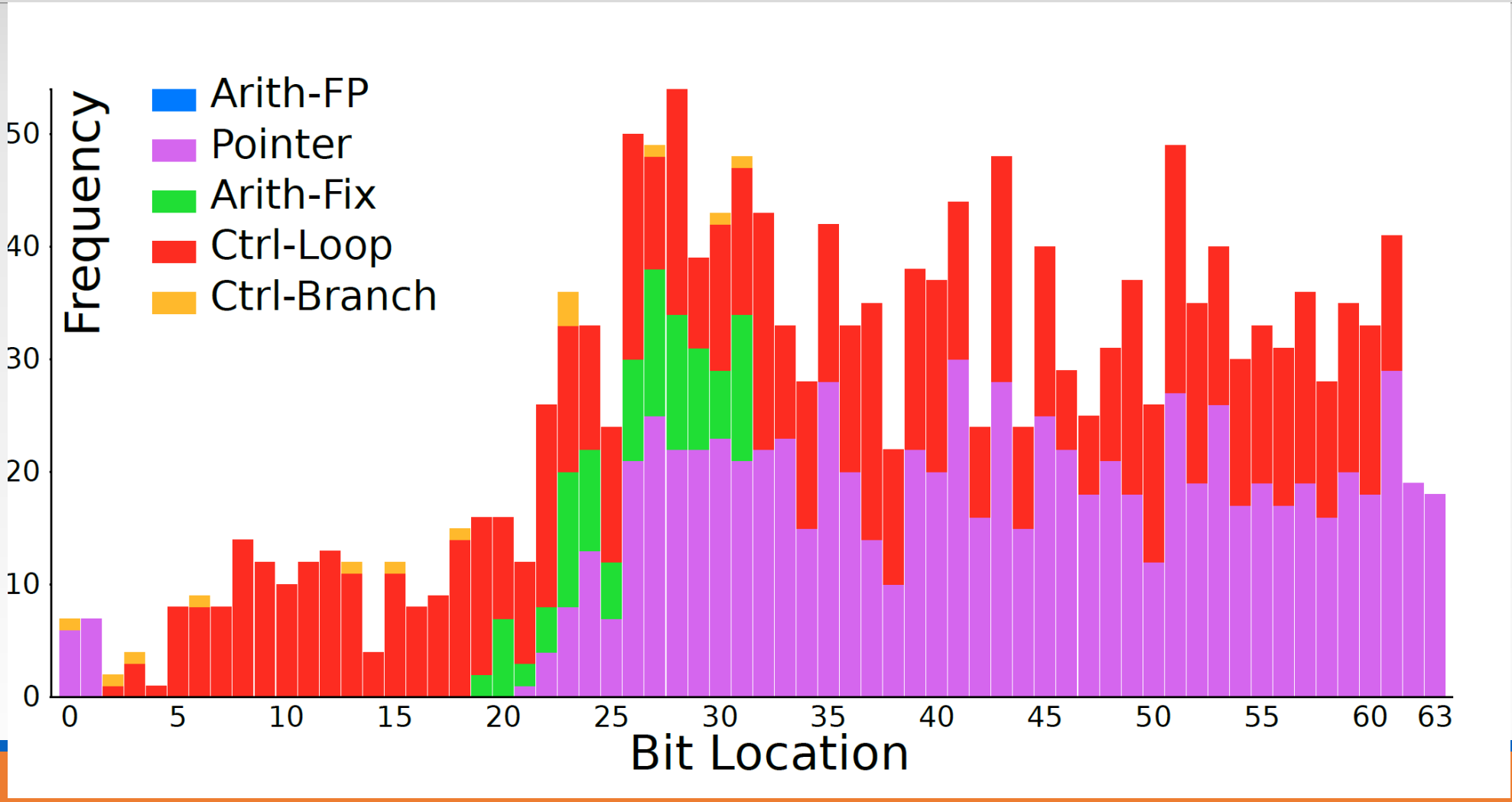
Bit locations that lead to longer runtime



Breakdown of injections by function



Injected bits that caused a segfault



HW3

You will use FlipIt to inject bit-flips into an HPC application on a supercomputer

You will use FaultSight to visualize the fault injection campaign and conduct analysis

Summary

Covered various methods of injecting fail-stop errors

Discussed transient error detectors at a low, medium, and high level

Overview of FlipIt and FaultSight that you will use on your next homework

References

DeBardeleben et al “SC 13 Tutorial: Practical Fault Tolerance on Today’s HPC Systems” LA-UR-13-26640

Schadenfreude: https://en.wikipedia.org/wiki/Pentium_FDIV_bug