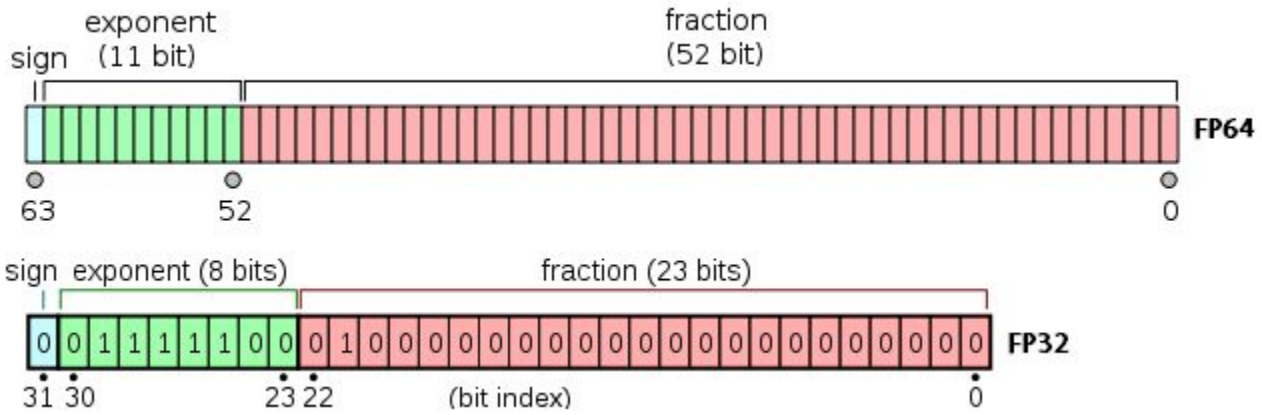
Survey of Tools to Assess Reduced Precision on Floating Point Applications

By Quinn Dibble

Project Mentors: **Terry Grové, Laura Monroe**
Supercomputer Institute 2020
ASC Beyond Moore's Law Inexact Computing
LA-UR-20-25935

August 6th, 2020

Motivation



- Floating point computation is a staple of scientific computing
- High precision is accurate, but has high energy, runtime, and resource costs
- Mixed precision is a way to offset some of those costs
  - This is the goal of the ASC BML inexact computing project

- **Manually figuring out mixed precision config is hard - tools?**

Image:
https://www.thecrazyprogrammer.com/wp-content/uploads/2018/04/Single-Precision-vs-Double-Precision.png

Overview

Six tools will be covered:

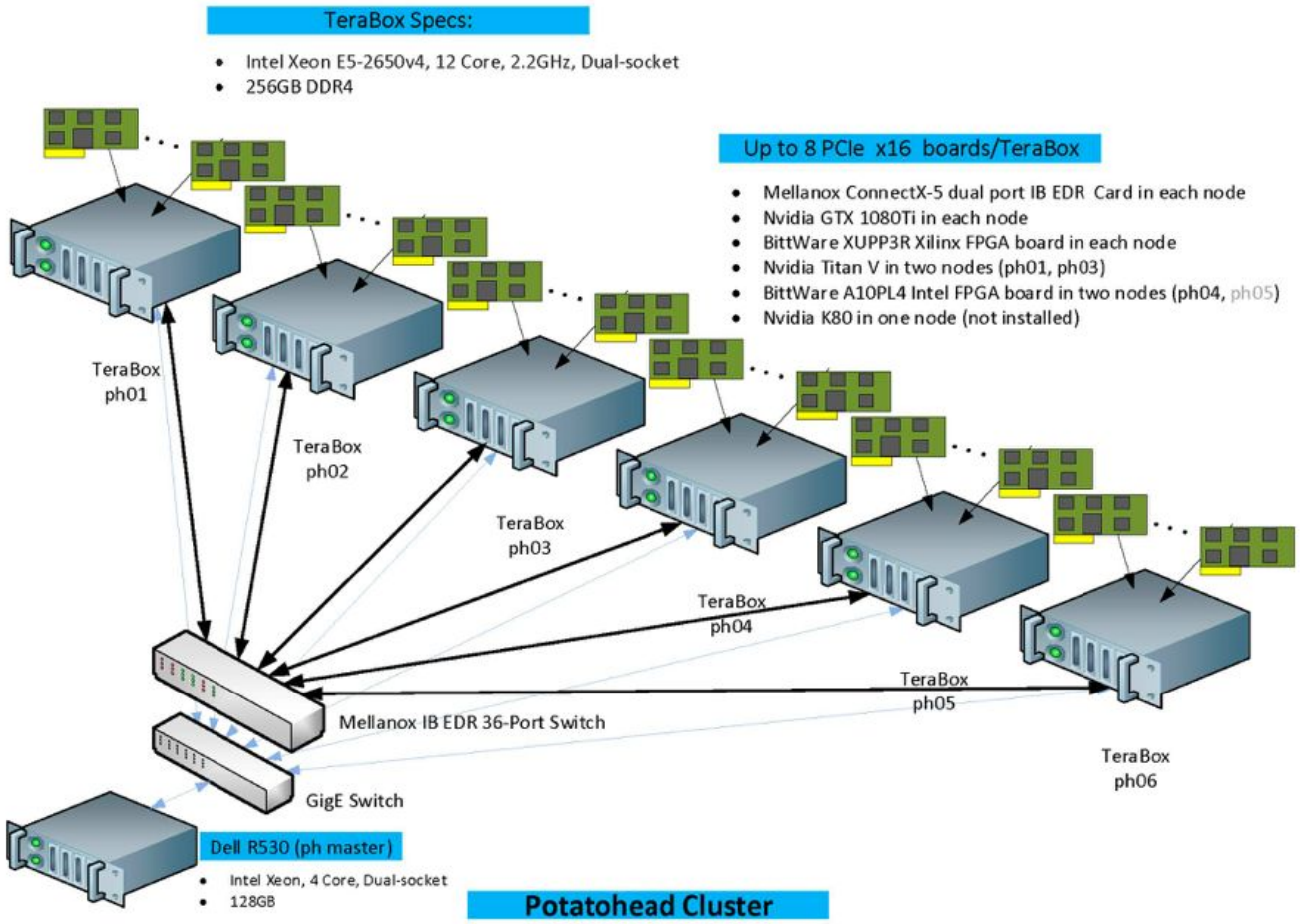
- ADAPT
- FLiT
- FloatSmith
- FPBench
- HiFPTuner
- Precimonious

Potatohead test system

- Small test cluster put together for ASC Beyond Moore's Law Inexact Computing project
- Flexible and incorporated cutting-edge devices
- Relevant to tools tests:
  - 2x Xeon E5-2623 4 core CPU @3GHz
  - 126G Memory, 1G swap

Image courtesy of Andy DuBois, HPC-DES

Potatohead schematic



ADAPT

Algorithmic Differentiation Applied to Floating Point Precision Tuning

Github: https://github.com/LLNL/adapt-fp
Paper: https://dl.acm.org/doi/10.5555/3291656.3291720

Harshitha Menon, Daniel Osei-Kuffuor, Markus Schordan, Scott Lloyd, Kathryn Mohror, Jeffrey Hittinger - LLNL Center for Applied Scientific Computing

Michael O. Lam - James Madison University

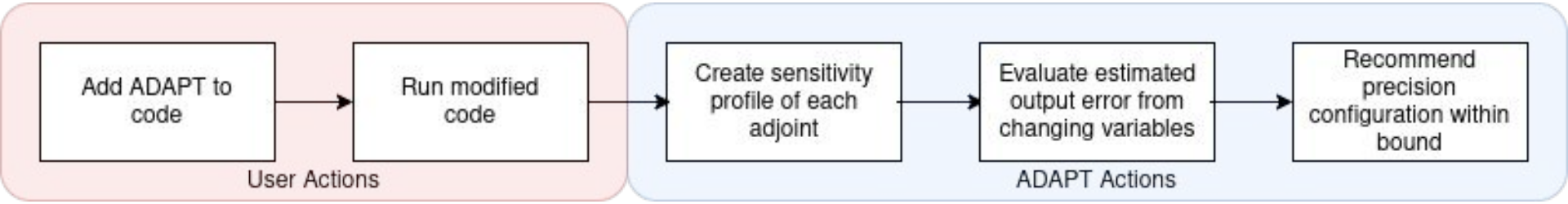
ADAPT - Overview

- C++ Library
- Find a lower precision version of your code within error bounds
- *Estimates error* caused by lowering precision

ADAPT - Usage

- Include adapt header files
- Change FP variables to AD_real type
- Tag independent, intermediate, and dependent variables with macros
- Use function calls to change analysis behavior

ADAPT - Workflow



ADAPT Tests

- Applied to publicly available mini-app CLAMR
- Added ADAPT code in a function to test
- Ate up so much RAM, OS killed it

```
AD_real AD_deltaT = deltaT;          AD_INDEPENDENT(AD_deltaT, "AD_deltaT");
AD_real AD_dr = dr;                  AD_INDEPENDENT(AD_dr, "AD_dr");
AD_real AD_U = U;                    AD_INDEPENDENT(AD_U, "AD_U");
AD_real AD_F_plus = F_plus;          AD_INDEPENDENT(AD_F_plus, "AD_F_plus");
AD_real AD_F_minus = F_minus;        AD_INDEPENDENT(AD_F_minus, "AD_F_minus");
AD_real AD_G_plus = G_plus;          AD_INDEPENDENT(AD_G_plus, "AD_G_plus");
AD_real AD_G_minus = G_minus;        AD_INDEPENDENT(AD_G_minus, "AD_G_minus");

AD_real AD_U_fullstep = (AD_U - (AD_deltaT / AD_dr)*(AD_F_plus - AD_F_minus + AD_G_plus + AD_G_minus));
AD_DEPENDENT(AD_U_fullstep, "AD_U_fullstep", 1e-7);
```

ADAPT - Conclusion

- Works well on very small scale - might be easier to tune manually?
- Can implement on single function/algorithm within code
- Not great for large scale programs:
  - Resource and time hog
  - Have to modify large codebase
- Straightforward to implement!

What if there was a more automated version of Adapt?

FloatSmith

Tool Integration for Source-Level Mixed Precision

Github: https://github.com/crafthpc/floatsmith

Paper: https://w3.cs.jmu.edu/lam2mo/papers/2019-Lam-Correctness.pdf

Tristan Vanderbruggen, Harshitha Menon, Markus Schordan - LLNL

Michael O. Lam - LLNL & James Madison University

Floatsmith - Overview

- Toolchain that leverages 3 tools:
  - TypeForge - find and replace variables
  - ADAPT (optional) - narrow search space
  - CRAFT - A tool to search and test different FP configs

FloatSmith - Overview

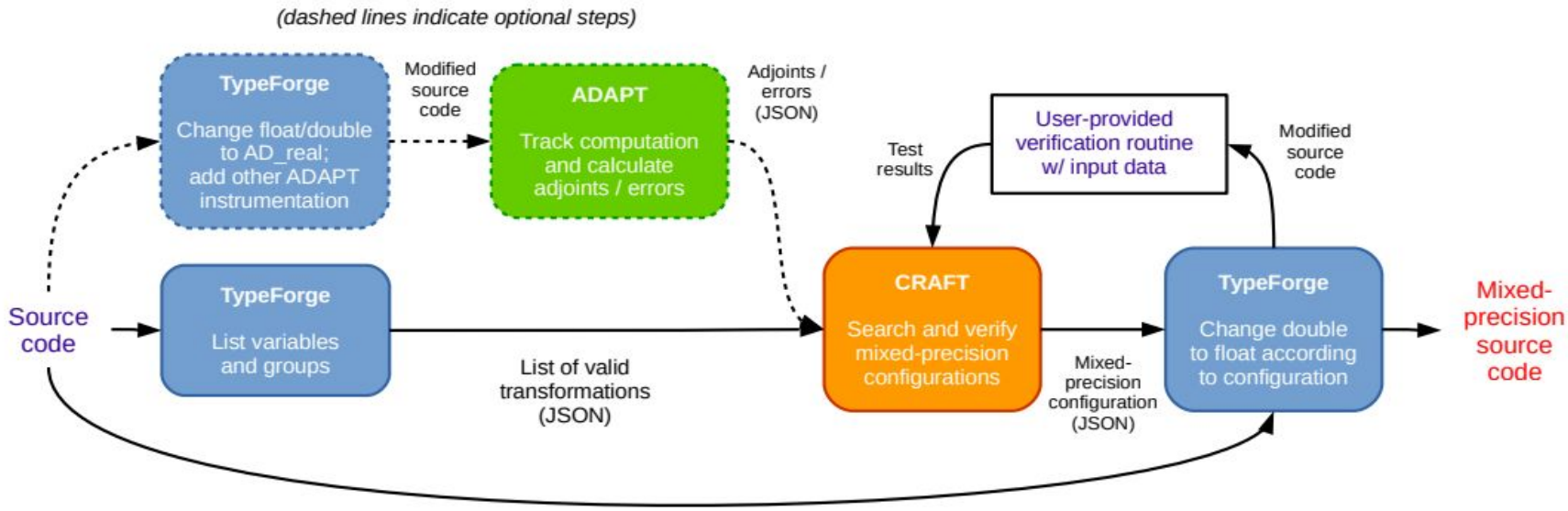


Fig. 1. Integrated system tool chain overview

Figure taken from paper: https://w3.cs.jmu.edu/lam2mo/papers/2019-Lam-Correctness.pdf

Floatsmith - Usage

- Interactive script, ask user how to:
  - Build the program
  - Run the program
  - Declare a configuration valid (error, output match)
- Batch mode exists for automation

Floatsmith Tests

- Tested examples in Floatsmith repository
  - Ran premade batch mode scripts: looked good
  - Ran interactive: results depended on choices (search algorithm)
- Tested Floatsmith on CLAMR
  - Asked different things than example
  - Couldn’t generate list of variables

FloatSmith Conclusions

- Very easy to use on small programs (inc. examples)
- Absolutely use it with smaller programs
- Difficult to get working for complex code bases
  - Possibly pull out an algorithm from bigger codebase?

Precimonious

Tuning Assistant for Floating-Point Precision

Github: https://github.com/corvette-berkeley/precimonious

Paper: https://web.cs.ucdavis.edu/~rubio/includes/sc13.pdf

Cindy Rubio-González, Cuong Nguyen, Hong Diep Nguyen, James Demmel, William Kahan, Koushik Sen - EECS Department, UC Berkeley

David H. Bailey, Costin Iancu - Lawrence Berkeley National Lab (LBL)

David Hough - Oracle Corporation

Precimonious - Overview

- Finds a lowest floating point configuration of code within error
- Utilizes LLVM bitcode for modifications
- Tests error by running every configuration in search space

Precimonious - Workflow

Usage

- Create search file (manually or script)
- Run search script
- Test against original code with user specified error bound

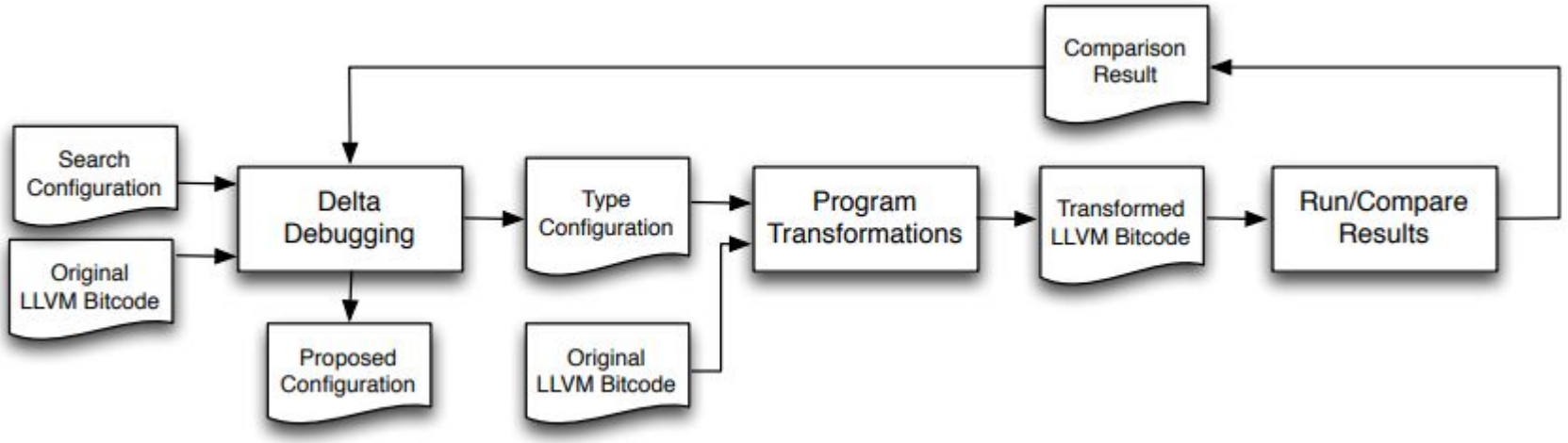


Image taken from Figure 3 in the paper: link

Precimonious Conclusions

- 6 year old project - might cause dependency issues with newer projects
- Not much in the documentation, only says how to install & run example
- Actually runs all configurations - large runtime costs

HiFPTuner

Exploiting Community Structure for Floating-Point Precision Tuning

Github: https://github.com/ucd-plse/HiFPTuner

Paper: https://web.cs.ucdavis.edu/~rubio/includes/issta18.pdf

Hui Guo, Cindy Rubio-González

Department of Computer Science - UC Davis

HiFPTuner - Overview

- An algorithm on top of Precimonious to improve search efficiency
- Still uses Precimonious for actual tuning

HiFPTuner - Approach

HiFPTuner approach:

1. Create LLVM bitcode file of program
2. Run analysis and transformation passes to attain dependence graph
3. Run Networkx and Community packages
4. Tune code with Precimonious

HiFPTuner - Conclusions

- Slightly faster search than Precimonious due to improved algorithm
- Have to change between Clang versions between steps
- If you really want to use Precimonious instead of FloatSmith/ADAPT, use this

FLiT

Cross-Platform Floating-Point Result-Consistency Tester and Workload

Github: https://github.com/PRUNERS/FLiT

Paper: https://ieeexplore.ieee.org/document/8167780

Geof Sawaya, Michael Bentley, Ian Briggs,
Ganesh Gopalakrishnan - University of Utah
Dong H. Ahn - LLNL

FLiT - Overview

- Test infrastructure to find variation in FP code caused by different factors:
  - Compilers
  - Compiler Optimizations
  - Hardware
  - Execution Environments

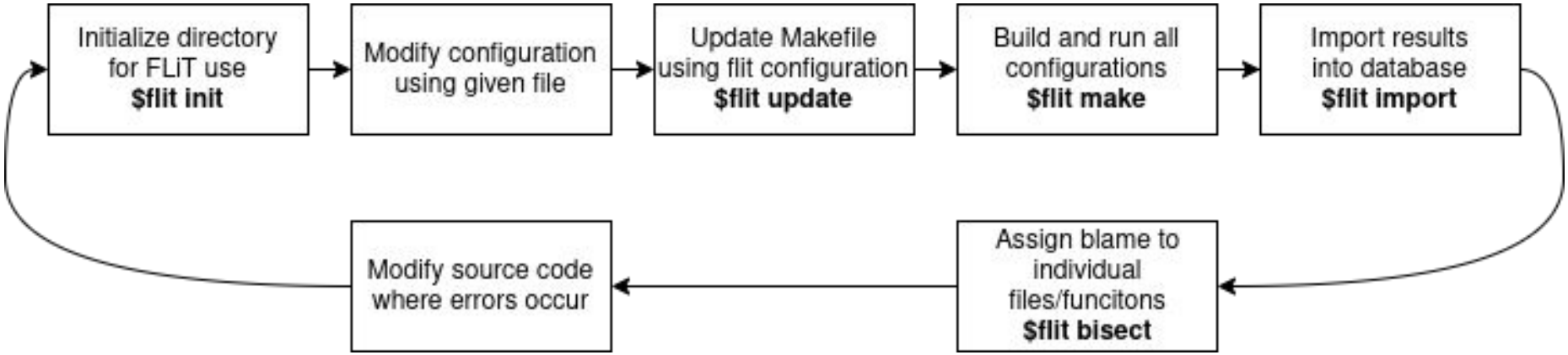
FLiT - Components

- C++ reproducibility test infrastructure
- dynamic make system
- SQLite database and analysis tools for results
- Bisection tool that can isolate file(s) and function(s) that introduce variability

FLiT - Approach

- Runs every combination of compiler(s) & optimizations
  - Compares results to “ground truth” - unoptimized run
  - Measures runtime
- Create database for results
- Comes with “litmus tests”
  - Tests that common FP algorithms
  - Tests designed to expose runtime/compiler behavior

FLiT - Workflow



FLiT - Test

- Ran “litmus-tests” with GCC and Clang, excluded intel compiler
- Took ~12 hours to compile and run all configurations
- Command line utility is very easy to use!

FLiT - Conclusions

- If you've finished your code, and want to test portability
- Must have your own "goodness metric" output
- Very good documentation

FPBench

Toward a Standard Benchmark Format and Suite for Floating-Point Analysis

Website: http://fpbench.org/index.html

Github: https://github.com/FPBench/FPBench

Nasrine Damouche, Matthieu Martel - Université de Perpignan Via Domita

Pavel Panchekha, Chen Qiu, Alexander Sanchez-Stern, Zachary Tatlock - University of Washington

FPBench - Overview

- A suite that provides benchmarks, compilers, and standards for FP research
- Includes FPCore format - standardized way to express FP algorithms

FPBench - Workflow

- Write algorithm in FPCore format
- Run transform tool:
  - Simplify preconditions
  - Unroll loops
  - Expand syntactic sugar
- Run export tool to convert FPCore to language like C

FPBench - Conclusions

- If you already have a written program, no tool to convert it to FPCore
- Not for using FP to research other topics
- For researching FP computation
  - Example: what happens if I have this FP equation with these conditions?

Conclusion

- All these tools can be useful, but are pretty niche
  - Expect to spend a decent chunk of time getting tools working with your code
- You are expected to know what results are “good”
- For precision tuning, I recommend starting with FloatSmith

Tool Reference

Tool	What it is	Use	Recommended When?
ADAPT	C++ library	Find mixed precision with fine control (e.g. just one algorithm)	Only when fine control is needed, small program/algorithm
FloatSmith	Interactive Toolchain	Interactive script to find mixed precision, fast checking on small program	When it works, easiest to get running → try this first Tune entire small program
Precimonious	Tool (scripts)	Find mixed precision version of code with float, double, long double precision	Only on small program that has long doubles - must be able to compile to LLVM
HiFPTuner	Tool (scripts)	Find mixed precision version of code with float, double, long double precision, improved search algorithm	Small projects, must be able to compile to LLVM bitcode
FLiT	Test infrastructure	Test reproducibility in different compilers/environments	If you have defined output, and hardware time
FPBench	Benchmarks + Standards	FP-specific research	Testing FP algorithms, haven't implemented actual code yet

Questions?

Acknowledgements:

Laura Monroe, Terry Grové - Mentors

Reid Priedhorsky - Director, Supercomputer Institute

ASC Beyond Moore's Law Inexact Computing Team