POLITECNICO
MILANO 1863

VALUE RANGE ANALYSIS AND FEEDBACK-DRIVEN OPTIMIZATION

FOR A MIXED PRECISION COMPILER

Daniele Cattaneo, Michele Chiari, Antonio Di Bello,
Stefano Cherubin, Giovanni Agosta

What is Precision Tuning?

Precision tuning is the process of **adjusting the precision of the variables** used in a program to improve its performance characteristics

Done through **numeric representation changes**

Example: small floating point → big floating point, or floating point → fixed point

Why Precision Tuning?

- Most benefits on slow CPUs
  - Application: Embedded Systems (most frequent application)
- Also shown to benefit fast CPUs
  - Application: High Performance Computing

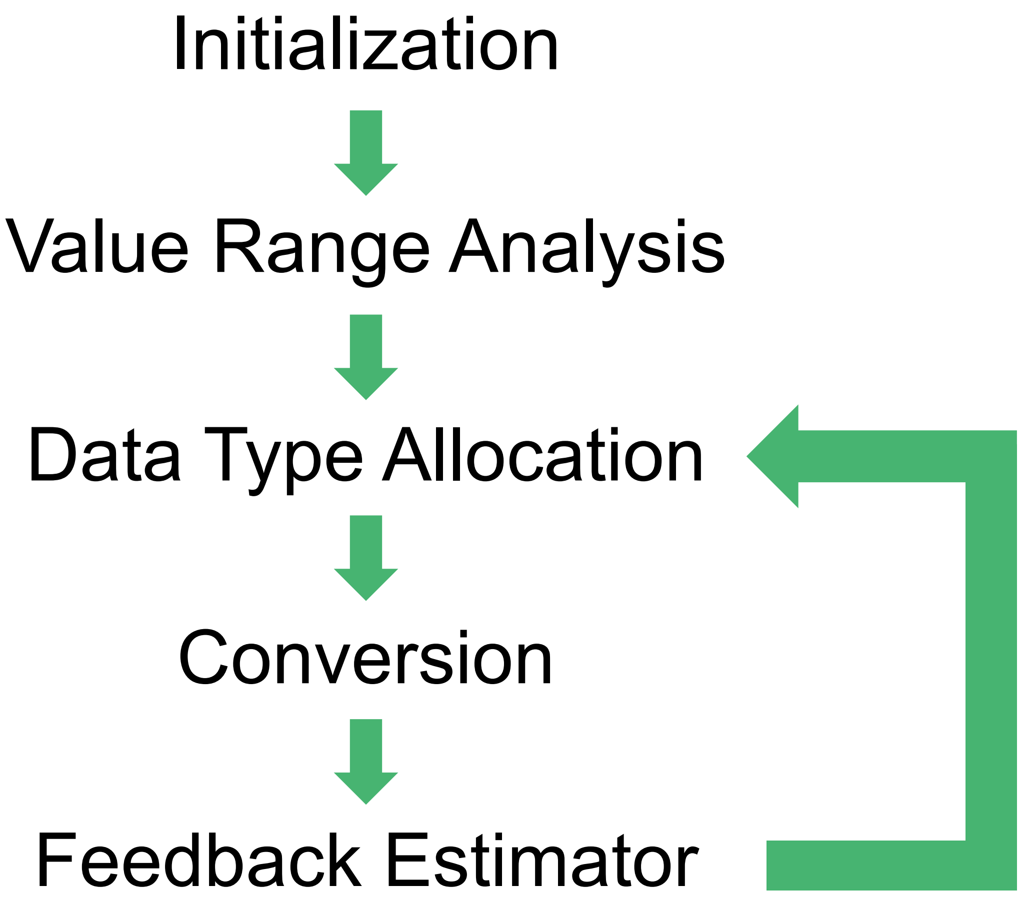
TAFFO: A New Mixed Precision Compiler

- TAFFO performs the whole precision tuning process
  - Using a state-of-the-art compiler (LLVM)
  - Incorporating state-of-the-art analyses
  - Focusing on floating-point to fixed-point
- Includes a complex code conversion module meant to operate on real-world code
- Modular and extensible

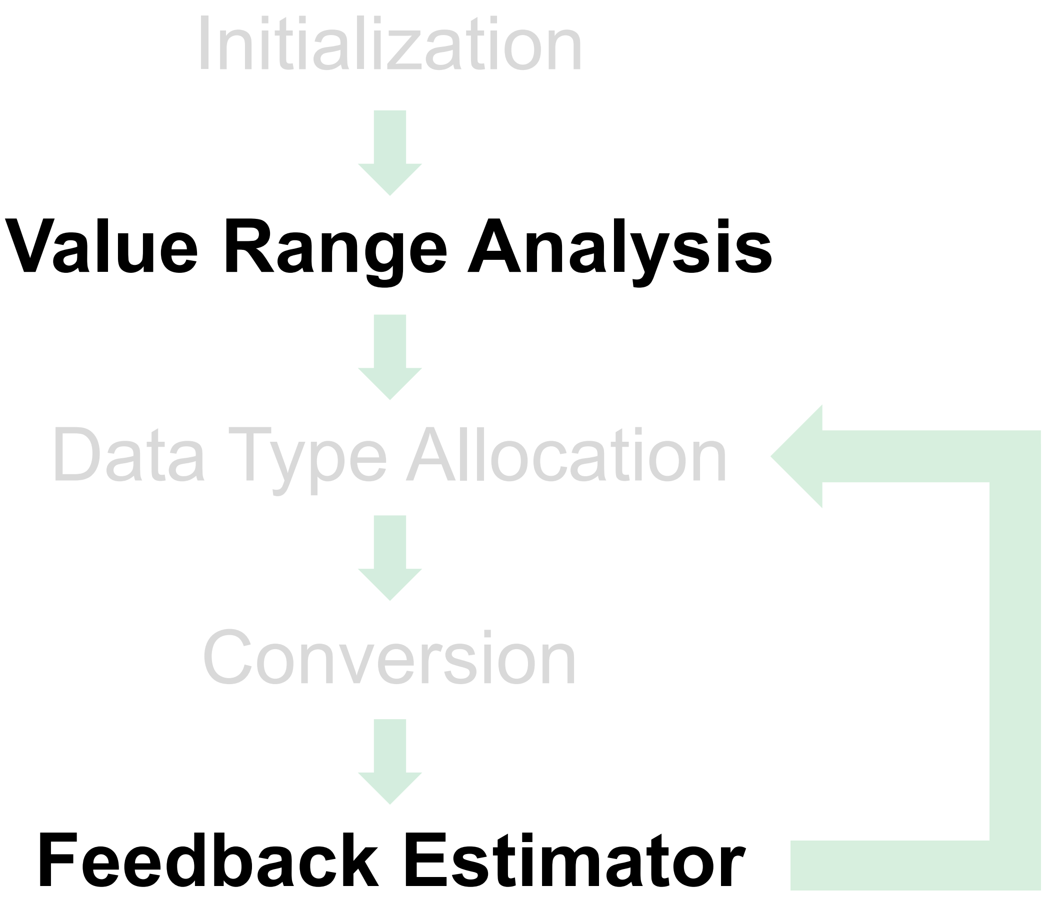
What is Handled

- Common arithmetic operations and comparisons
  - *add*, *sub*, *mul*, equal-to, greater-than, ...
- Memory operations
  - Arrays and pointers included
- PHI nodes, Select
- Constants
- Global Variables
- Library Functions
- Non-library Functions

Architecture of TAFFO



Architecture of TAFFO



Value Range Analysis

```
float i, j = 0;
```

[0, 0.9]   [0, 4.17]

```
for (i=0; i<1; i+=0.1)  {
      j = j + sin(i);
}
```

- Uses a state-of-the-art methodology
- Based on Range Arithmetic

Value Range Analysis Algorithm

- Symbolic execution of the program using Range Arithmetic for the values
- In case of loop
  - Estimate loop trip count (via LLVM Scalar Evolution)
  - Simulate loop body that number of times OR until the symbolic values reach a fixed point

Before and after...

- Before VRA:
  - One annotation per variable, everywhere
  - Bugs in intermediate values due to inappropriate precision choices, requiring manual tweaks
  - Less type casts, **more speedup**

- After VRA:
  - Annotation of only a few key variables
  - The optimized code works out of the box
  - More type casts, **less speedup**...

The speedup loss must be regained somehow!

Feedback Estimator

1. Estimates error on user selected variables
2. Machine learning model to estimate performance
   - Metric: Instruction Mix
   - Metric: Amount & kind of code changes made by TAFFO
3. Automatically change TAFFO behavior based on collected data

Performance Estimation Metrics

- # of instructions
- # of instructions affected by TAFFO
- loop depth
- trip count
- relative instruction mix with & without TAFFO

Performance Estimation Model

- Choice of the model based on experimentation
- Best option: Gradient Tree Boosting Classification
- Classification System:

    -1: slowdown

    0: no improvement

    +1: speedup

Error Propagation

- Symbolic execution of the program using Affine Arithmetic for computing the errors at each instruction
- In case of loop (just like VRA)
  - Estimate loop trip count
  - Simulate loop body that number of times OR until the symbolic values reach a fixed point

- Always conservative!

Feedback!

- User choice: prefer low error or high performance?
- Low error:
  - User provides a maximum error bound
  - "Precision parameter" is lowered until error reaches bound
  - If speedup classification is -1, do not use TAFFO, otherwise success!
- High performance:
  - Same thing but symmetric

Precision Parameter?

1. Every fixed point type gets a score (= size of frac. part + size of int. part)
2. for all instructions
   – if instruction uses different types
     - if difference between scores < **threshold**, change types to the type with largest integer part

- The score threshold (Q) is the "precision parameter"

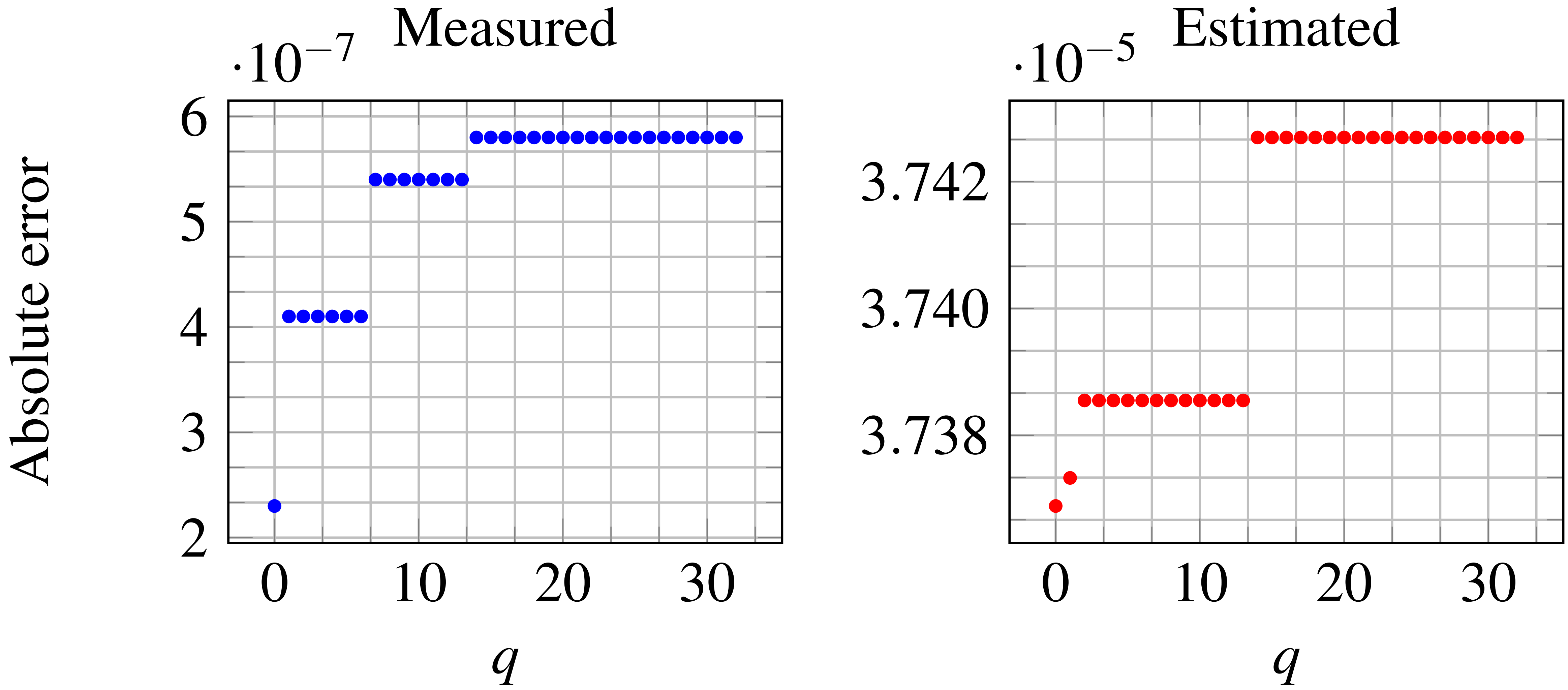
Dataset

- *PolyBench/C*
  - Collection of **micro-kernels**
- *AxBench*
  - Collection of **applications** for approximate computing research
    - Financial Analysis (Black-Scholes)
    - Signal Processing (FFT)
    - Robotics (Inversek2j)
    - 3D gaming (Jmeint)
    - Machine Learning (K-means)
    - Image Processing (Sobel)

Experiments & Issues

- 98% accuracy in training (Polybench)
- 100% accuracy in production (AxBench)

- Suspiciously good...
- Need more data but code isn't cheap to collect



Black-Scholes from AxBench

Conclusion

- Even a rough VRA is enough to make real-world applications work
- Data shows that optimization based on feedback on Q is a sound idea
- Performance estimation based on machine learning needs more time in the oven

Question time