ACCEPT: A PROGRAMMER-GUIDED COMPILER FRAMEWORK FOR PRACTICAL APPROXIMATE COMPUTING

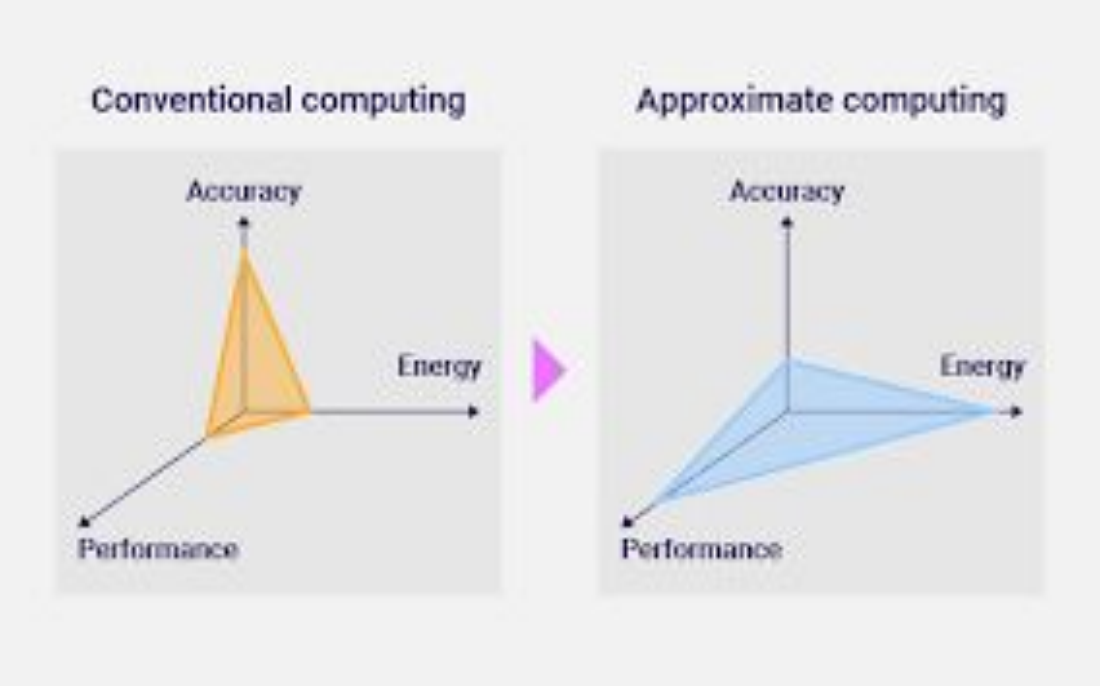
Presented by Group 7:
Arjun Laxman, Dhanush Kumar Mallu, Dimash Umirbayev, Gabriela Ribeiro Znamensky

Overview

- Approximate Computing - Introduction and Motivation
- What is ACCEPT?
- ACCEPT Workflow
  - Programmer-Compiler Feedback
  - Analysis Library
    - Precise-Pure Analysis
    - Approximate Region Selection
  - Autotuning
- Experimental Setup
- Experimental Results
- Strengths and Weaknesses

Introduction: Approximate Computing

- Many workloads can tolerate reduced accuracy:
  - Graphics / Image processing
  - Machine Learning
  - Browser Search Results
  - Physics simulations
- Approximate computing trades accuracy for faster runtimes and increased energy efficiency



https://co-design.t.u-tokyo.ac.jp/wp-content/uploads/2020/03/research_approximate2.png

Introduction: Approximate Computing Techniques

Examples:

- Loop Perforation
- Floating Point Precision Reduction
- Memoization / Lookup Tables
- Fast Inverse Square Root Algorithm

```
for (int i = 0; i < N; i++) {
    // do things
    i = i + skip_factor;
}
```

https://en.wikipedia.org/wiki/Loop_perforation

```
float Q_rsqrt( float number )
{
    long i;
    float x2, y;
    const float threehalfs = 1.5F;

    x2 = number * 0.5F;
    y  = number;
    i  = * ( long * ) &y;
    i  = 0x5f3759df - ( i >> 1 );
    y  = * ( float * ) &i;
    y  = y * ( threehalfs - ( x2 * y * y ) );
//  y  = y * ( threehalfs - ( x2 * y * y ) );

    return y;
}
```

https://en.wikipedia.org/wiki/Fast_inverse_square_root

Motivation: Approximate Computing Challenges

- Manually applying approximate computing techniques is tedious, error-prone, and requires a strong understanding of the code

- Fully automated approximate computing sacrifices programmer control and visibility into program behavior

- To balance these approaches, we need …

… ACCEPT

Approximate C Compiler for Energy and Performance Trade-offs

- Static analysis based on programmer intent
- Dynamic testing of real program behavior
- Combined approach enables controlled and practical approximation

ACCEPT Workflow

- Programmer-Compiler Feedback based on type qualifier annotations and analysis logs
- Analysis Library performs precise-pure analysis and approximate region selection
- Autotuning System explores relaxation variations to determine optimal configurations

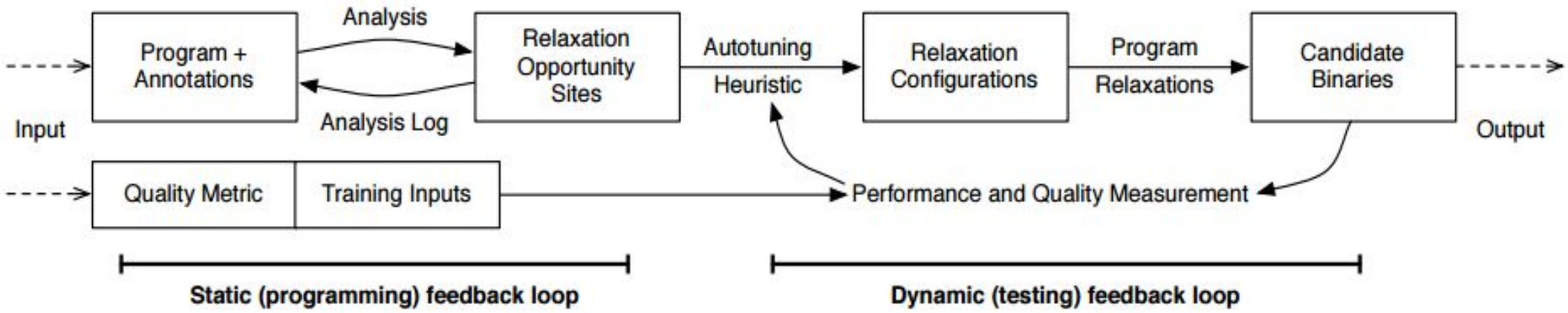


Figure 1: Overview of the ACCEPT compiler workflow.

Programmer-Compiler Feedback

- Types are either precise (default) or approximate (marked with APPROX)
  - Information Flow
    - Approximate data can't impact precise data unless explicitly through ENDORSE
    - Impacts pointers and control flow
  - Escape Hatches (ACCEPT_PERMIT, ACCEPT_FORBID) override compiler decisions on a statement's effects
- Implemented through annotations passed to IR from modified Clang
- Analysis log lists all attempted relaxations and if it's safe/what makes it unsafe

```
APPROX int a = expensiveCall();
cheapChecksumPrecise(a); // illegal
```

```
APPROX int a = expensiveCall();
cheapChecksumPrecise(ENDORSE(a));
```

Analysis Library: Precise-Pure Analysis

- Basis for determining if a given relaxation is applicable
  - A region is precise pure if it
    - Has no stores to precise data reachable outside the region
    - Only calls precise-pure functions
    - Has no unbalanced synchronization
- Based on def-use chains and pointer-escape analysis
  - Implemented as an analysis pass in LLVM

```
int p = … ;

APPROX int a = p * 2;
```

Analysis Library: Approximate Region Selection

- Identifies candidate regions that can be offloaded to accelerators
  - A candidate region
    - Is precise-pure
    - Has single entry/single exit control flow
    - Has identifiable live-ins/live-outs
- Based on dominator/post-dominator analysis
  - Implemented as an analysis pass in LLVM

Algorithm 1: Approximate region selection.

Input: function f
Output: *set of precise-pure regions R in f*

```
1  foreach basic block B in f do
2      foreach block B′ strictly post-dominated by B do
3          if B′ dominates B then
4              region ← formRegionBetween(B′, B)
5              if region is precise-pure then
6                  R ← R ∪ {region}
7              end
8          end
9      end
10 end
```

Autotuning

- Exhaustively checking relaxation configurations is infeasible even for moderately sized programs
- Heuristic Process
  - Relaxations are vetted individually to discard unsafe or unhelpful options
  - Configurations are ranked by estimated performance-to-quality ratio
    - Quality is programmer-specified and application-specific
  - Run candidate configurations to measure true speedup and output error
  - Generates a Pareto-optimal set
- Integrates with LLVM compiler through command-line flags and a configuration file

Experimental Setup

- Evaluation Platforms
  - Intel Xeon server - distributed autotuning on 20 processors
  - Zynq SoC with FPGA - used as a neural-network accelerator
  - MSP430 Microcontroller - embedded sensor platform
- Optimizations applied to benchmark programs
  - Loop Perforation - only run every *p*th loop iteration
  - Synchronization Elision - remove lock/unlocks and barriers
  - Neural acceleration - approximable code is offload to neural network

Experimental Setup

Application	Description	Quality Metric
canneal	VLSI routing	Routing cost
fluidanimate	Fluid dynamics	Particle distance
streamcluster	Online clustering	Cluster center distance
x264	Video encoding	Structural similarity
sobel	Sobel filter	Mean pixel difference
zynq-blackscholes	Investment pricing	Mean relative error
zynq-inversek2j	Inverse kinematics	Euclidean distance
zynq-sobel	Sobel filter	Mean pixel difference
msp430-activity	Activity recognition	Classification rate

Experimental Results

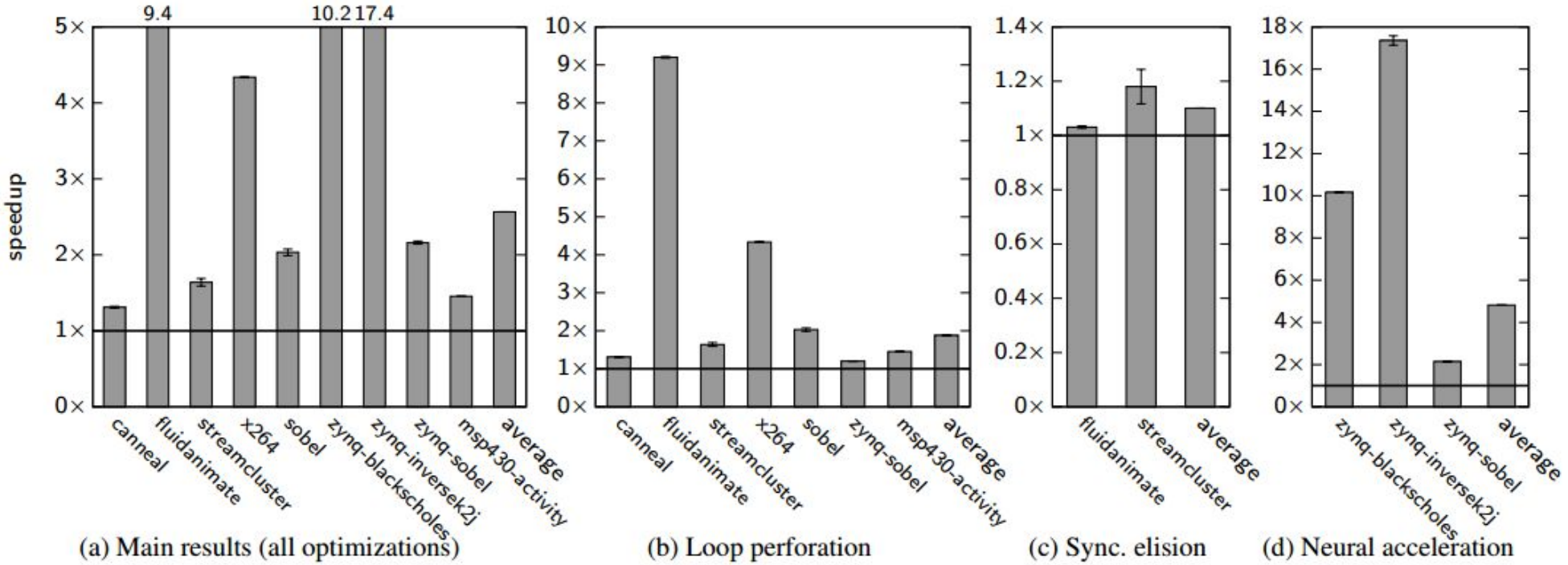


Figure 3: Speedup for each application, including all optimizations (a) and each optimization in isolation (b–d).

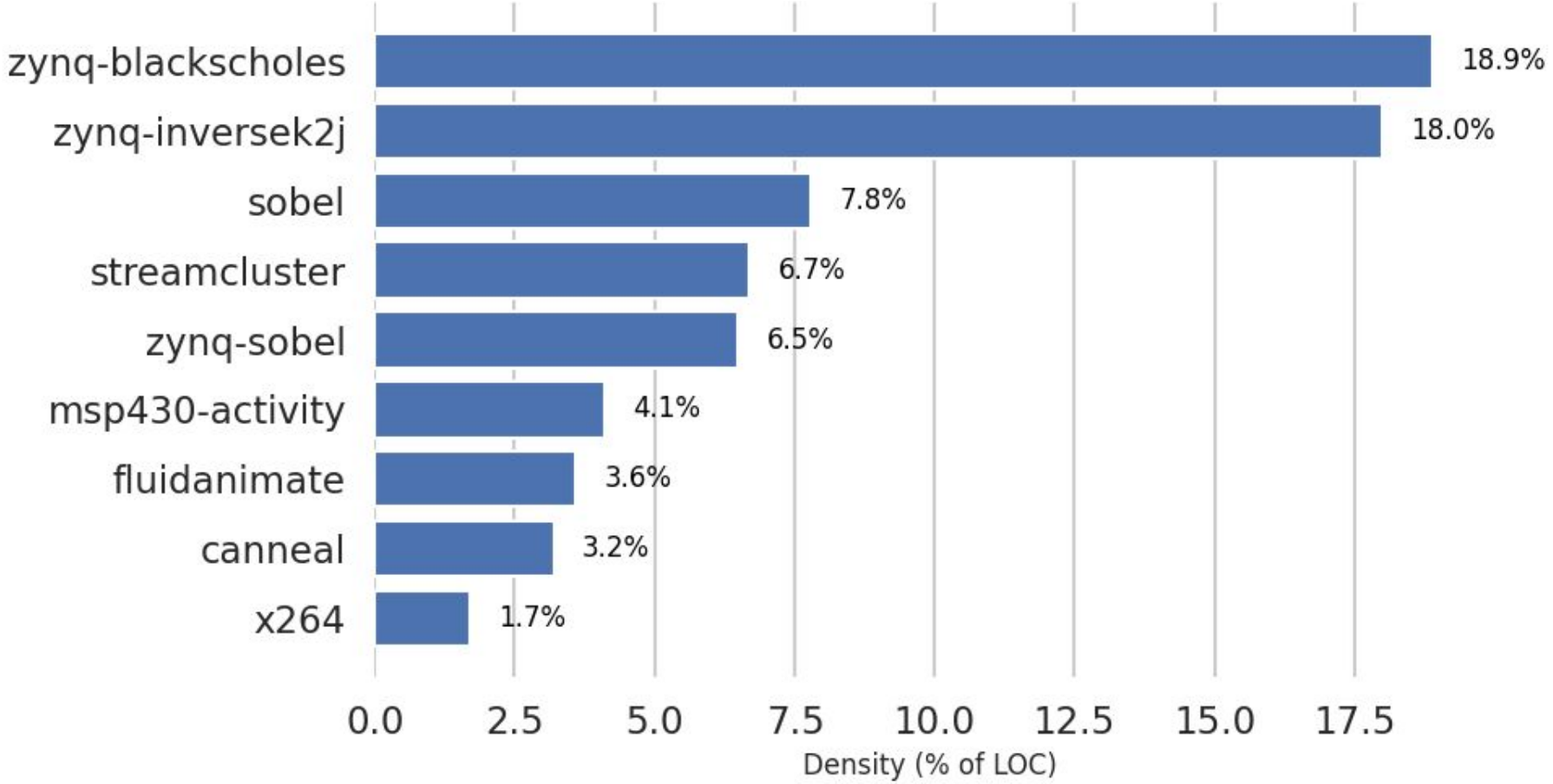
Experimental Results

- Average Speedup per Platform
  - Intel Xeon server - **2.3x**
  - Zynq SoC with FPGA - **4.8x**
  - MSP430 Microcontroller - **1.5x**
- Other considerations
  - Other Pareto-Optimal Designs
  - Optimization time
  - Energy consumption
  - Annotation overhead

Application	Sites	Composites	Total	Optimal	Error	Speedup
canneal	5	7	32	11	1.5–15.3%	1.1–1.7×
fluidanimate	20	13	82	11	<0.1%	1.0–9.4×
streamcluster	23	14	66	7	<0.1–12.8%	1.0–1.9×
x264	23	10	94	3	<0.1–0.8%	1.0–4.3×
sobel	6	5	21	7	<0.1–26.7%	1.1–2.0×
zynq-blackscholes	2	1	5	1	4.3%	10.2×
zynq-inversek2j	3	2	10	1	8.9%	17.4×
zynq-sobel	6	2	27	4	2.2–6.2%	1.1–2.2×
msp430-activity	4	3	15	5	<0.1%	1.5×

Table 2: Tuning statistics and resulting optimal configurations for each benchmark.

Annotation Overhead



Strength and Weakness

STRENGTHS	WEAKNESS
Manual/Automatic Balance	Conservative Analysis
Safe Analysis	Linear Composition
Flexible Framework	Limited Scope
Practical Results	Type Safety Limits
Accessibility	Escape hatch trade-offs

Thank you!

Questions?