Towards Statistical Guarantees in Controlling Quality Tradeoffs for Approximate Acceleration

Divya Mahajan
Amir Yazdanbakhsh
Jongse Park
Bradley Thwaites
Hadi Esmaeilzadeh

Alternative Computing Technologies (ACT) Lab
Georgia Institute of Technology
Approximate Computing

Relax the abstraction of "near perfect" accuracy in

- Data Processing
- Storage
- Communication

Accept *imprecision* to improve
  - performance
  - energy efficiency
Approximate Acceleration

Approximate acceleration enables this tradeoff
Approximate Acceleration
Fixed Error

Lack of Error Control

Lack of Guarantees
Fixed Error

Lack of Error Control

Lack of Guarantees
Fixed Error
Lack of Error Control
Lack of Guarantees

3D diagram showing a Pareto Frontier with axes for Error, Performance, and Energy.
To tackle these shortcomings we devise MITHRA.
Overview

Motivation

Challenges in devising MITHRA

A hardware software solution

Detailing the components of MITHRA
Motivation

Exploiting Accelerator Characteristics

Application

Approximate Accelerator

Final Output
Challenges

How to eliminate anomalous invocations?

Application

Approximate Accelerator

Local Error

Final Quality loss
Factors Influencing Error

Accelerator Error = |Output_{accelerator} - Output_{original}|

Output_{accelerator} = f (accelerator inputs, accelerator configuration)
Factors Influencing Error

Accelerator Error = | Output_{accelerator} - Output_{original} |

Output_{accelerator} = f (accelerator inputs, accelerator configuration)

Constant
Factors Influencing Error

Accelerator Error = \mid \text{Output}_{\text{accelerator}} - \text{Output}_{\text{original}} \mid

\text{Output}_{\text{accelerator}} = f (\text{accelerator inputs, accelerator configuration})

\text{Output}_{\text{accelerator}} = f (\text{accelerator inputs})
Challenges

How to eliminate anomalous invocations?

Look at the accelerator inputs
Challenges

Approximate Accelerator

Local Error

Final Quality loss

Final Quality loss → Local Error?
Challenges

How to eliminate anomalous invocations?

Look at the accelerator inputs

Final Quality loss \rightarrow Local Error?

Threshold the local error
Challenges

- How to eliminate anomalous invocations?
- Final Quality loss → Local Error?
- What guarantees?

- Look at the accelerator inputs
- Threshold the local error
Challenges

How to eliminate anomalous invocations?

Look at the accelerator inputs

Final Quality loss → Local Error?

Threshold the local error

What guarantees?

Statistical Guarantees
Challenges

- How to eliminate anomalous invocations?
- Final Quality loss → Local Error?
- What guarantees?
- What algorithm at runtime?

Look at the accelerator inputs
- Threshold the local error
- Statistical Guarantees
Challenges

How to eliminate anomalous invocations?

Look at the accelerator inputs

Final Quality loss → Local Error?

Threshold the local error

What guarantees?

Statistical Guarantees

What algorithm at runtime?

Classification in Hardware
MITHRA: A Hardware/Software Solution

Approximate Accelerator

Desired quality requirements

Generate the threshold for the local error such that the final quality loss meets the requirements

Input Datasets

Application

Statistical Optimizer

Desired quality requirements

Generate the threshold for the local error such that the final quality loss meets the requirements

Input Datasets
MITHRA: A Hardware/Software Solution

---

Approximate Accelerator

Application

Desired quality requirements

Statistical Optimizer

Classifier Trainer

Input Datasets

Generate training data that segregates inputs that give > and < (th)
MITHRA: A Hardware/Software Solution

Desired quality requirements

Input Datasets

Hardware Classifier Topology
**MITHRA: A Hardware/Software Solution**

Approximate Accelerator

Desired quality requirements

Statistical Optimizer (th)

Classifier Trainer

Hardware Classifier Topology

Input Datasets
Statistical Optimizer

Input Datasets

Application

Approximate Accelerator
Statistical Optimizer

Approximate Accelerator

Precise Result

Input Datasets

Application
Statistical Optimizer

Input Datasets

Application

Approximate Accelerator

Approximate Result
Statistical Optimizer

Input Datasets

Application

Approximate Accelerator

Local error
Statistical Optimizer

Input Datasets

Application

Approximate Accelerator

Local error

\( \geq \)

Threshold
Local error → th → Application
Approximate Accelerator

Desired Quality Loss by the programmer

Input Datasets

Application

Final Quality loss
Binomial Proportion Confidence Interval

- Less than the desired programmer quality loss
  \[ \text{nsuccess} \]

- Greater than the desired programmer quality loss

Input Datasets
**Binomial Proportion Confidence Interval**

Less than the desired programmer quality loss

Greater than the desired programmer quality loss

\[
\frac{1}{1 + \frac{(n_{\text{trials}} - n_{\text{success}} + 1)}{n_{\text{success}} \times F[1 - \alpha; 2n_{\text{success}}, 2(n_{\text{trials}} - n_{\text{success}} + 1)]}} < \text{SuccessRate}
\]

with a confidence level
Example

\[(n_{\text{trials}}, n_{\text{success}}) \Rightarrow r < \text{SuccessRate} \]

with a confidence level

E.g., \((100, 80) \Rightarrow 72.28\% < \text{SuccessRate} \]

with 95\% confidence level
Example

\[(n_{\text{trials}}, n_{\text{success}}) \implies r < \text{SuccessRate} \quad \text{with a confidence level}\]

E.g., \((100, 80) \implies 72.28\% < \text{SuccessRate} \quad \text{with 95\% confidence level}\]

Final Quality Level, Success Rate and Confidence Interval programmer specified
if desired metrics are not met:
\[ th_{t+1} = th_t - \Delta \]
else if desired metrics are met:
\[ th_{t+1} = th_t + \Delta \]

Reiterate; till the \( th_t \) meets the metrics but \( th_{t+1} \) doesn't
Statistical Optimization

if desired metrics are not met:
\[ th_{t+1} = th_t - \Delta \]
else if desired metrics are met:
\[ th_{t+1} = th_t + \Delta \]

Reiterate; till the \( th_t \) meets the metrics but \( th_{t+1} \) doesn't

Tighter threshold better Final Quality Loss Level, Success Rate and Confidence Interval but lower benefits from approximation
Training the Classifiers

The training data used to generate classifier topology
Hardware Classifiers

Simple algorithm that can be easily implemented in hardware.

We use two techniques for this work:
1. Table Based
2. Neural Network Based
Table-based Classifiers

Accelerator Inputs → Classifier → Input Signature → Precise

Multiple Input Signature Register (MISR):

<table>
<thead>
<tr>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
A small **ensemble** of table-based classifiers achieve better accuracy and performance.
Neural Network Based Classifiers

Input Layer → Hidden Layer → Output Layer

in0 → in1 → in2 → in3 → in4 → in5 → in6 → in7 → in8

Hidden Layer

→ '0'

→ '1'
## Benchmarks

<table>
<thead>
<tr>
<th></th>
<th>Accelerator Topology</th>
<th>Baseline Error</th>
<th>Neural Classifier Topology</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blackscholes</td>
<td>6 → 8 → 8 → 1</td>
<td>6.02%</td>
<td>6 → 4 → 2</td>
</tr>
<tr>
<td>FFT</td>
<td>1 → 4 → 4 → 2</td>
<td>7.22%</td>
<td>1 → 4 → 2</td>
</tr>
<tr>
<td>Inversek2j</td>
<td>2 → 8 → 2</td>
<td>7.50%</td>
<td>2 → 4 → 2</td>
</tr>
<tr>
<td>JMEINT</td>
<td>18 → 32 → 8 → 2</td>
<td>17.69%</td>
<td>18 → 16 → 2</td>
</tr>
<tr>
<td>JPEG Encoding</td>
<td>64 → 16 → 64</td>
<td>7.00%</td>
<td>64 → 2 → 2</td>
</tr>
<tr>
<td>Sobel</td>
<td>9 → 8 → 1</td>
<td>9.96%</td>
<td>9 → 4 → 2</td>
</tr>
</tbody>
</table>

Table Classifier Topology: 8 tables each of size 0.5 KB
Energy and Performance Benefits

![Graphs showing energy and speedup benefits with varying application quality loss.

For each application, there are three lines representing different approaches:

- **Oracle**
- **Table-based**
- **Neural**

As the application quality loss increases, the speedup and energy reduction also increase. The Oracle method consistently outperforms the other two in terms of both speedup and energy reduction.

The graphs illustrate that the Neural approach shows improved performance compared to the Table-based method, especially at higher application quality losses.

Overall, these results highlight the potential benefits of using Neural approaches in terms of both energy efficiency and performance.
Invocation Rate

![Graph showing the relationship between Application Quality Loss and Invocation Rate for different methods: Oracle, Table-based, and Neural. The graph includes markers for each method at different quality loss rates.]
False Positive and False Negative Results

![Graph showing false negative and false positive results for table-based and neural methods vs application quality loss.](image_url)
Varying Success Rate

Improvement in Energy-Delay Product

Success Rates

- 75%
- 80%
- 85%
- 90%
- 95%
- 97%

Oracle | Table-based | Neural
Conclusion

MITHRA

Hardware software co-design works well

Aims to make statistical guarantees a norm
Thank you