CS 598sm
Probabilistic & Approximate Computing

http://misailo.web.engr.Illinois.edu/courses/cs598
High quality, High cost

Medium quality, Medium cost

Low quality, Low cost

High quality, High cost
High quality, High cost

Medium quality, Medium cost

Low quality, Low cost

Quality

Cost (Time and Energy)

0%

100%
High quality, High cost

Medium quality, Medium cost

Low quality, Low cost

High quality, High cost
Find an approximate program

Various automatic or user-guided approaches
ACCURACY MATTERS
Precision

Repeatability or fineness of control

Less precise  More precise

Accuracy
Difference from the correct value

Reliability

Probability that a system has been functioning correctly, continuously over the time interval \([0, t]\)

Conventionally denoted by the function \(R(t)\)

Sometimes we implicitly use without \(t\), meaning that reliability is over the period of operation

Another Thought Experiment

What if we change magnitude of the pixel?
What if we change frequency of the pixel (sometimes it’s just black)?
Function’s and Program’s Accuracy

Magnitude of Noise

Difference $d$ between the exact and approximate pixel values (for all color components)
Function’s and Program’s Accuracy

**Frequency of Noise**

Probability $p$ with which interpolation kernel produces a correct pixel
Accuracy Requirement

**Specify Metric and Threshold**

- Each application has its own
- Require domain expertise (the only part!)
- For visual data, historically often PSNR is used
- But one can think of other better perceptory metrics

---

**Definition**

PSNR is most easily defined via the mean squared error (MSE). Given a noise-free $m \times n$ monochrome image $I$ and its noisy approximation $K$, $MSE$ is defined as:

$$MSE = \frac{1}{mn} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} [I(i,j) - K(i,j)]^2$$

The PSNR (in dB) is defined as:

$$PSNR = 10 \cdot \log_{10} \left( \frac{MAX_I^2}{MSE} \right)$$

$$= 20 \cdot \log_{10} \left( \frac{MAX_I}{\sqrt{MSE}} \right)$$

$$= 20 \cdot \log_{10} (MAX_I) - 10 \cdot \log_{10} (MSE)$$

Here, $MAX_I$ is the maximum possible pixel value of the image. When the pixels are represented using 8 bits per sample, this is 255. More generally, when samples are represented using linear PCM with $B$ bits per sample, $MAX_I$ is $2^B - 1$.

---

More details on the roles of metrics: Karpuzcu et al., On Quantification of Accuracy Loss in Approximate Computing, WDDD 2015.
Accuracy Requirement

*Specify Metric and Threshold*

![Graph showing PSNR vs. Acceptable Quality](image)
Accuracy Specifications

End-to-end: program output
- You can compare outputs only at the end of the run
- Often better understood for representative domains

Kernel-level: each function has its specification
- Fine-grained control + checking of intermediate results
- Often ad-hoc or not intuitive
- While in general can lead to composition, hard to propagate all errors
Accuracy Requirement

Specify Metric and Threshold

Interpolation Reliability

PSNR

Acceptable Quality
Analytic Derivation

Use properties of the algorithm and implementation

Local Specification
- Pixel kernel reliability \( r \)

Global Specification
- PSNR of the image

Computation Pattern
- Data parallel loop

\[
PSNR(D, D') = 20 \cdot \log(255) - 10 \cdot \log \left( \frac{1}{h \cdot w} \sum_{i,j} (D_{ij} - D'_{ij})^2 \right)
\]

\[
r \cdot 0 + (1 - r) \cdot 255
\]
Analytic Derivation

*Use properties of the algorithm and implementation*

Local Specification: Pixel kernel reliability $r$

Global Specification: PSNR of the image

Computation Pattern: Data parallel loop

$$\mathbb{E}[\text{PSNR}(D, D')] \geq -10 \cdot \log(1 - r)$$
Accuracy-Aware Optimization

- Find an approximate program
- Apply transformations that change semantics

Original Computation ✔

Accuracy Requirement ✔

Optimized Computation +
Loop Perforation (2009)

for (i = 0; i < n; i++) { ... }

for (i = 0; i < n; i += 2) { ... }

Misailovic, Sidiroglou, Hoffmann, Rinard Quality of Service Profiling (ICSE 2010)
Sidiroglou, Misailovic , Hoffmann, Rinard Managing Performance vs. Accuracy Trade-offs With Loop Perforation (FSE 2011)
Loop Perforation

```plaintext
for (i = 0; i < n; i++) { ... }
```

```plaintext
for (i = 0; i < n/2; i++) { ... }
```
Loop Perforation

```c
for (i = 0; i < n; i++) { ... }

for (i = 0; i < n; i++) {
    if (rand(0.5)) continue;
    ...
}
```
Reduction Sampling

for (i = 0; i < n; i++) {
    y = f(x[i]);
    s = s + y;
}

for (i = 0, z = 0; i < n; i++) {
    if (rand(0.75)) {z++; continue;}
    y = f(x[i]);
    s = s + y;
}

s = s * n/(n-z);

Zhu et al. Randomized Accuracy-Aware Program Transformations For Efficient Approximate Computations, POPL ‘12
Approximate Memoization

InType[] x; OutType[] y;
for (i = 0; i < n; i++) { y[i] = f(x[i]); }

var table = new Map<InType, OutType>;
for (i = 0; i < n; i++) {
    if ∃x',v . x'∈[x[i]-ε, x[i]+ε] && (x',v)∈table
        y[i] = v;
    else {
        y[i] = f(x[i]);
        table[x[i]] = y[i];
    }
}
Approximate Tiling

```c
InType[] x; OutType[] y;
for (i = 0; i < n; i++) {
    y[i] = f(x[i]);
}
```

```c
InType prev;
for (i = 0; i < n; i++) {
    if (i%2 == 1)
        y[i] = prev;
    else {
        y[i] = f(x[i]);
        prev = y[i];
    }
}
```

Chaudhuri et al. Proving Programs Robust, FSE ‘11
Samadi et al., Paraprox Pattern-Based Approximation for Data Parallel Applications, ASPLOS’14
Image Perforation: Automatically Accelerating Image Pipelines by Intelligently Skipping Samples, SIGGRAPH’16
Function Substitution

\[ y = f(x); \]

\[ y = f'(x); \]

<table>
<thead>
<tr>
<th>Version</th>
<th>TimeSpec</th>
<th>ErrorSpec</th>
</tr>
</thead>
<tbody>
<tr>
<td>( f(x) )</td>
<td>Time1</td>
<td>Err1</td>
</tr>
<tr>
<td>( f'(x) )</td>
<td>Time2</td>
<td>Err2</td>
</tr>
</tbody>
</table>

For instance, polynomial approximation of transcendental functions:

\[
\sin(x) \approx x - \frac{x^3}{3!} + \frac{x^5}{5!} - \cdots \text{ for } x \text{ near } 0
\]

\[
R(x) \leq |x|^{n+1} / (n + 1)!
\]

Baek et al., PLDI 10; Ansel et al., CGO ’11
Function Substitution

\[ y = f(x); \]

\[ y = f'(x); \]

<table>
<thead>
<tr>
<th>Version</th>
<th>TimeSpec</th>
<th>ErrorSpec</th>
</tr>
</thead>
<tbody>
<tr>
<td>( f(x) )</td>
<td>Time1</td>
<td>Err1</td>
</tr>
<tr>
<td>( f'(x) )</td>
<td>Time2</td>
<td>Err2</td>
</tr>
</tbody>
</table>

Neural Network:

Esmaeilzadeh et al., Neural Acceleration for General-Purpose Approximate Programs, MICRO '12
Dynamic Function Substitution

\[ y = f(x); \]

\[ y = \text{runtime.executeApprox}()? f'(x): f(x); \]

<table>
<thead>
<tr>
<th>Version</th>
<th>TimeSpec</th>
<th>ErrorSpec</th>
</tr>
</thead>
<tbody>
<tr>
<td>(f(x))</td>
<td>Time1</td>
<td>Err1</td>
</tr>
<tr>
<td>(f'(x))</td>
<td>Time2</td>
<td>Err2</td>
</tr>
</tbody>
</table>

- Baek et al., Green: A Framework for Supporting Energy-Conscious Programming using Controlled Approximation, PLDI 2010
- Hoffmann et al., Dynamic Knobs for Efficient Power Aware Computing, APSLOS 2011
- Mitra et al., Phase-aware Approximation in Approximate Computing, CGO 2017
Floating Point Optimizations

double[] x, y
double z = f(x,y)

float[] x, y
float z = f(x,y)

Rubio-Gonzalez et al., Precimonious: Tuning Assistant for Floating-Point Precision, SC 2013
# Skipping Tasks (at Barrier Points)

<table>
<thead>
<tr>
<th>task</th>
<th>task</th>
<th>task</th>
<th>task</th>
<th>task</th>
<th>task</th>
</tr>
</thead>
<tbody>
<tr>
<td>`{</td>
<td>`{</td>
<td>`{</td>
<td>`{</td>
<td>`{</td>
<td>`{</td>
</tr>
<tr>
<td>x = ...</td>
<td>x = ...</td>
<td>x = ...</td>
<td>x = ...</td>
<td>x = ...</td>
<td>x = ...</td>
</tr>
<tr>
<td>y = ...</td>
<td>y = ...</td>
<td>y = ...</td>
<td>y = ...</td>
<td>y = ...</td>
<td>y = ...</td>
</tr>
</tbody>
</table>
| }     | }     | }     | }     | }     | }

Continue execution after all tasks finish

Continue execution after all tasks finish **before timeout,**
Otherwise kill delayed or non-responsive tasks

---

Rinard, Probabilistic accuracy bounds for fault-tolerant computations that discard tasks, ICS '06
Meng et al. Best-Effort Parallel Execution for Recognition and Mining Applications, IPDPS'09
Removing Synchronization

lock();
x = f(x,y);
y = g(x,y);
unlock();

lock();
x = f(x,y);
y = g(x,y);
unlock();

Renganarayana et al. Programming with Relaxed Synchronization, RACES ‘12
Misailovic et al. Dancing with Uncertainty, RACES ‘12
Transformations

Dimensions of impact:

• **Reducing computation**
  (perforation, memorization, tiling, function substitution)

• **Reducing data**
  (floating point optimizations)

• **Reducing communication/synchronization**
  (skipping tasks and lock elision)
Applying Transformations

Selecting \textbf{where} in code to approximate

- **Programmer-guided:** programmer writes annotations
- **Automatic:** system identifies the code and tunes the approximation
- **Combined:** programmer writes some annotations, system infers the rest
- **Interactive:** system identifies the code and presents the results to the developer who accepts/rejects
Applying Transformations

Choosing the level of approximation:

• Off-line: before execution starts
• On-line: during execution
• Combined: improve off-line models with on-line data
Some Key Characteristics:

• **Approximate Kernel Computations**
  (have specific structure + functionality)

• **Accuracy vs Performance Knob**
  (tune how aggressively to approximate kernel)

• **Magnitude and Frequency of Errors**
  (kernels rarely exhibit large output deviations)