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)
Accuracy-Aware Optimization

- Find an approximate program
- Various automatic or user-guided approaches
ACCURACY ~ CORRECTNESS
Precision

Repeatability or fineness of control

Accuracy

Difference from the correct value

More accurate

Less accurate

Less precise

More precise

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 that interpolation kernel produces (for all color components)
Function’s and Program’s Accuracy

Frequency of Noise

Probability $p$ with which interpolation kernel produces the correct pixel
We observe

Small Errors

Most of the Time
Accuracy Requirement

Specify Metric and Threshold

• Each application has its own
• Requires domain problem expertise
• For visual data, historically PSNR has often been used (with all its imperfections)
• 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×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
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

![Graph showing PSNR vs Interpolation Reliability with Acceptable Quality threshold]
Analytic Derivation

*Use properties of the algorithm and implementation*

Local Specification: Kernel computes the pixel with 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

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

Perforated

> 1% pixel difference
Perforated

> 5% pixel difference
x264 Motion Estimation

Reference Frame

Current Frame
x264 Block Matching

score = 0;

for (i = 0; i < block_height; i++) {
    for (j = 0; j < block_width; j++) {

        idx1 = IDX(i, j, cur_start);
        idx2 = IDX(i, j, prev_start);
        diff = cur_frame[idx1] - prev_frame[idx2];
        adif = abs(diff);
        score = score + adif;
    }
}

return score;
score = 0;

for (i = 0; i < block_height; i+=2) {
    for (j = 0; j < block_width; j+=2) {
        idx1 = IDX(i, j, cur_start);
        idx2 = IDX(i, j, prev_start);
        diff = cur_frame[idx1] - prev_frame[idx2];
        adif = abs(diff);
        score = score + adif;
    }
}

return score;
score = 0;

for (i = 0; i < block_height; i+=2) {
    for (j = 0; j < block_width; j+=2) {

        idx1 = IDX(i, j, cur_start);
        idx2 = IDX(i, j, prev_start);
        diff = cur_frame[idx1] - prev_frame[idx2];
        adif = abs(diff);
        score = score + adif;

    }
}

return score * 4;
Absolute Error of Perforation

*With Bias Compensation*

Most of the time errors of individual approximation computations are small!
Several Patterns Amenable to Approximation

- Map
- Reduce (sum, average, min, max, median)
- Stencil
- Scatter/Gather
- Iterative refinement loop
- ...
Accuracy-Aware Optimization

- **Find** an approximate program
- **Apply** transformations that change semantics

Optimized Computation
Tradeoffs

![Graph showing tradeoffs between quality and cost.](image)

- **Quality** axis ranging from 0% to 100%
- **Cost (Time and Energy)** axis
- **Optimization System** indicating a point on the graph

---

*Figure 1: Graph illustrating the tradeoffs between quality and cost for a system.*
Transformations and induce a space of approximate executions

Many of these executions will be similar to the original execution

We want their final results to be similar (i.e., low accuracy loss)

Ideally, we want the execution that runs the fastest

Sometimes, we can enforce that approximate programs must always execute near the original.

It can help the analysis, but is not necessary.
General Optimization Problem

Select Program Configuration $X \in \text{Configs}$ to

\[
\text{maximize } (\text{Speedup}(X, i), \text{Accuracy}(X, i)) \\
\text{forall } i \in \text{InputSet}
\]

But these are most often competing objectives.

Rephrase: for every accuracy loss threshold $\delta$

\[
\text{maximize } \text{Speedup}(X, i) \\
\text{subject to } \text{AccuracyLoss}(X, i) \leq \delta \\
\text{forall } i \in \text{InputSet}
\]
Multiobjective Optimization

Functions to optimize are called *objectives*

- Accuracy Loss – lower is better (or accuracy – higher is better)
- Speedup – higher is better (or normalized time – lower is better)
- Energy saving – higher is better (or consumption – lower is better)

They are the functions of program configuration – setting of knobs

Two candidate program configurations X and Y:

- X *Pareto dominates* Y if X is as good as Y in all objectives, and is better in at least one objective

*Pareto frontier:* the set of points that are not dominated by other points

*We will come back and formalize these notions later in the course!*
Example

![Graph showing the relationship between Speedup and Accuracy Loss]

- Speedup
- Accuracy Loss

- Data points indicating a trend or pattern in the relationship between Speedup and Accuracy Loss.
Example
Example

![Graph showing Speedup vs. Accuracy Loss with a scatter plot.](image-url)
Example

Speedup vs. Accuracy Loss

Data points are scattered, showing a trade-off between speedup and accuracy loss.
Example

![Graph showing the relationship between Speedup and Accuracy Loss](image)
Example
Example

Pareto (non-dominated) front

[Graph with axes labeled Speedup and Accuracy Loss, showing a Pareto front with blue and orange dots connected by a line]
Example

True Pareto front (theoretical optimum)

Pareto front

Speedup

Accuracy Loss
Pareto Fronts (aka Tradeoff curves)

- Convex
- Non-Convex
- Discontinuous
- Concave
Spread of Solutions:

Often to have a useful set of points, a developer would like to have points spread across the entire space, not located only at the corners.
Accuracy-Aware Optimization

- Find an approximate program
- Apply transformations that change semantics

Optimized Computation +
SOFTWARE TRANSFORMATIONS
Transformations

Dimensions of impact:
• Reducing computation

• Reducing data

• Reducing communication/synchronization
Floating Point Optimizations

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

float[] x, y
float z = f(x, y)
Speedup = \frac{\text{Original program time}}{\text{Approximate program time}}

<table>
<thead>
<tr>
<th>Program</th>
<th>10^{-10}</th>
<th>10^{-8}</th>
<th>10^{-6}</th>
<th>10^{-4}</th>
</tr>
</thead>
<tbody>
<tr>
<td>arclength</td>
<td>41.7%</td>
<td>41.7%</td>
<td>11.0%</td>
<td>33.3%</td>
</tr>
<tr>
<td>simpsons</td>
<td>13.7%</td>
<td>7.1%</td>
<td>37.1%</td>
<td>37.1%</td>
</tr>
<tr>
<td>bessel</td>
<td>0.0%</td>
<td>0.0%</td>
<td>0.0%</td>
<td>0.0%</td>
</tr>
<tr>
<td>gaussian</td>
<td>0.0%</td>
<td>0.0%</td>
<td>0.0%</td>
<td>0.0%</td>
</tr>
<tr>
<td>roots</td>
<td>6.8%</td>
<td>6.8%</td>
<td>4.5%</td>
<td>7.0%</td>
</tr>
<tr>
<td>polyroots</td>
<td>0.0%</td>
<td>0.0%</td>
<td>0.0%</td>
<td>0.0%</td>
</tr>
<tr>
<td>rootnewt</td>
<td>0.5%</td>
<td>1.2%</td>
<td>4.5%</td>
<td>0.4%</td>
</tr>
<tr>
<td>sum</td>
<td>0.0%</td>
<td>0.0%</td>
<td>0.0%</td>
<td>15.0%</td>
</tr>
<tr>
<td>fft</td>
<td>0.0%</td>
<td>0.0%</td>
<td>13.1%</td>
<td>13.1%</td>
</tr>
<tr>
<td>blas</td>
<td>0.0%</td>
<td>0.0%</td>
<td>24.7%</td>
<td>24.7%</td>
</tr>
<tr>
<td>ep</td>
<td>-</td>
<td>33.2%</td>
<td>32.3%</td>
<td>32.8%</td>
</tr>
<tr>
<td>cg</td>
<td>4.6%</td>
<td>2.3%</td>
<td>0.0%</td>
<td>15.9%</td>
</tr>
</tbody>
</table>
Loop Perforation

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

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

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;  
    ...  
    ...  
}
```
Managing Performance vs. Accuracy Trade-offs With Loop Perforation FSE 2011
Reduction Sampling

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

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

Zhu et al. Randomized Accuracy-Aware Program Transformations For Efficient Approximate Computations, POPL '12
Tradeoff curve for the main component of Bodytrack
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

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

InType prev;
for (i = 0; i < n; i++) {
    if (i%2 == 1)
        y[i] = prev;
    else {
        y[i] = f(x[i]);
        prev = y[i];
    }
}
Figure 15: The impact of approximate memoization on four functions on a GPU. Two schemes are used to handle inputs that do not map to precomputed outputs: nearest and linear. Nearest chooses the nearest value in the lookup table to approximate the output. Linear uses linear approximation between the two nearest values in the table. For all four functions, nearest provides better speedups than linear at the cost of greater quality loss.
Figure 1. Execution time breakdown of all PARSEC 3.0 benchmarks that LLVM could compile. The AVG column presents the average breakdown across all benchmarks. The AVG (eval.) column presents the average breakdown across the benchmarks we consider in the remainder of this study (which exclude bodytrack, freqmine, and canneal, which have almost no pure or extended pure functions). Pure functions cover a small fraction of the total execution time, while extended pure functions achieve significantly higher coverage.

Figure 4. TAF-Memo distortion versus relative runtime. TAF-Memo achieves significant speedups with small distortion for most applications.
Fig. 8. Image perforation and loop perforation results for four image pipelines from top to bottom: bilateral filter, bilateral grid, blur, demosaic, median and unsharp mask. Each row compares optimized pipelines computed using each method for similar speedup factors. Please consult the supplemental document for extensive comparisons for each of these pipelines. Note that one can zoom in to see the Bayer mosaic pattern for the demosaic input. From top to bottom row, credits: © Charles Roffey, Trey Ratcliffe, Neal Fowler, Eric Wehmeyer, Duncan Harris, Sandy Glass.
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:

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

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

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

\[ y = f(x); \]

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

\[
\begin{array}{|c|c|c|}
\hline
\text{Version} & \text{TimeSpec} & \text{ErrorSpec} \\
\hline
f(x) & \text{Time1} & \text{Err1} \\
f'(x) & \text{Time2} & \text{Err2} \\
\hline
\end{array}
\]

Neural Network:

Esmaeilzadeh et al., Neural Acceleration for General-Purpose Approximate Programs, MICRO ‘12
Figure 6: Cumulative distribution function (CDF) plot of the applications’ output error. A point \((x, y)\) indicates that \(y\) fraction of the output elements see error less than or equal to \(x\).

The Parrot transformation degrades each application’s average output quality by less than 10%, a rate commensurate with other approximate computing techniques.
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
Dynamic Approximation

swaptions

Power Cap:
Clock drops
2.4-1.6GHz

Power Cap lifted:
Clock rises 1.6-2.4 GHz

During the power cap, we either restart or suffer through poor performance.
Dynamic Approximation

swaptions

Application switches to the alternative implementation

Application returns to the original implementation
Skipping Tasks *(at Barrier Points)*

```plaintext
task {
  x = ...
  y = ...
}
```

Continue execution after all tasks finish

```plaintext
task {
  x = ...
  y = ...
}
```

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

\[
\text{lock();}
\]
\[
x = f(x, y);
\]
\[
y = g(x, y);
\]
\[
\text{unlock();}
\]

\[
\text{lock();}
\]
\[
x = f(x, y);
\]
\[
y = g(x, y);
\]
\[
\text{unlock();}
\]

\[
\text{lock();}
\]
\[
x = f(x, y);
\]
\[
y = g(x, y);
\]
\[
\text{unlock();}
\]

Renganarayana et al. Programming with Relaxed Synchronization, RACES '12
Misailovic et al. Dancing with Uncertainty, RACES '12
<table>
<thead>
<tr>
<th>Transformation</th>
<th>Speedup (max 8)</th>
<th>Relative Speedup</th>
<th>Accuracy Loss</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original</td>
<td>6.21</td>
<td>1.00</td>
<td>0.000 ± 0.000</td>
</tr>
<tr>
<td>BarrierInterf</td>
<td>6.34</td>
<td>1.02</td>
<td>0.027 ± 0.082</td>
</tr>
<tr>
<td>BarrierPoteng</td>
<td>6.48</td>
<td>1.04</td>
<td>0.035 ± 0.032</td>
</tr>
<tr>
<td>LockForces</td>
<td>6.34</td>
<td>1.02</td>
<td>0.004 ± 0.001</td>
</tr>
</tbody>
</table>

Table 1. Empirical Results for Individual Transformations

<table>
<thead>
<tr>
<th>Transformation</th>
<th>Speedup (max 8)</th>
<th>Relative Speedup</th>
<th>Accuracy Loss</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original</td>
<td>6.21</td>
<td>1.00</td>
<td>0.000 ± 0.000</td>
</tr>
<tr>
<td>BarrierInterf + LockForces</td>
<td>6.44</td>
<td>1.03</td>
<td>0.027 ± 0.044</td>
</tr>
<tr>
<td>BarrierPoteng + LockForces</td>
<td>6.79</td>
<td>1.09</td>
<td>0.042 ± 0.033</td>
</tr>
<tr>
<td>BarrierInterf + BarrierPoteng</td>
<td>7.10</td>
<td>1.14</td>
<td>0.053 ± 0.063</td>
</tr>
<tr>
<td>All Three</td>
<td>7.44</td>
<td>1.20</td>
<td>0.051 ± 0.070</td>
</tr>
</tbody>
</table>

Table 2. Empirical Results for Combinations of Transformations
Transformations

Dimensions of impact:

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

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

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

Selecting where in the 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 time to do the approximation:

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

We will discuss the algorithms and systems that help with approximating programs in detail!