CS 521: Topics in PL

Probabilistic & Approximate Computing

http://misailo.web.engr.illinois.edu/courses/cs521
Zoo:

SYSTEMS FOR ACCURACY-AWARE OPTIMIZATION
Accuracy-Aware Optimization

- **Find** an approximate program
- **Various** automatic or user-guided approaches
Testing-based Optimization

- **Transform** original computation
- **Validate** transformed computation

Optimized Computation +
For all inputs

Original Computation

Typical Inputs

Accuracy Requirement

---

**Analysis-driven Compiler**

SAS ‘11, POPL ’12, OOPSLA’13, OOPSLA ’14, OOPSLA’19, CGO ‘20

• **Statically analyze** computation’s accuracy

• **Transform** computation by solving a mathematical optimization problem

---

Optimized Computation +
Background: Compiler Autotuning

Search for program with maximum performance by reordering instructions, compiler parameters, and program configurations

- There are so many ways to tile an array (e.g., fit different cache sizes)
- Which optimizations to try –O1, -O2, -O3, remove some, add some?

Empirical process: explores the complexity of the system stack:

- Try new configuration
- If better then previous, save; and
- Search for more profitable configuration

Interesting educational project: https://github.com/ctuning/ck/wiki/Compiler-autotuning
Compiler Autotuning

Try new configuration: select one combination out of the space of all possible combinations
• Often too large to try them all
• The results will depend on the inputs you used

If better: (traditionally) compare performance or energy
• Uses fitness function which orders the configurations

Search for more: various heuristic algorithms, these days mainly based on machine learning and heuristic search (e.g., genetic programming in OpenTuner)

A Survey on Compiler Autotuning using Machine Learning (CSUR 2019)
Compiler Autotuning

Accuracy opens up a new dimension for search

- Increases the number of options to try
- Includes (input-specific) accuracy metric in the fitness fun.
- Finds the configurations with best tradeoffs.
Multiobjective Optimization (Reminder)

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 lecture!*
Example

![Speedup vs. Accuracy Loss Plot]

- Speedup
- Accuracy Loss

The plot shows a scatter of data points representing the relationship between speedup and accuracy loss.
Example
Example

Pareto (non-dominated) front
Example

True Pareto front (theoretical optimum)

Pareto front

Speedup

Accuracy Loss
Pareto Fronts (aka Tradeoff curves)

- Convex
- Non-Convex
- Discontinuous
- Concave
A BIT OF FORMALISM

Based on Knowels, Thiele, Zitzler
Optimization Problem

Optimization Problem is a Quadruple \((X, Z, f, \preceq)\):

- \(X\): decision space, and \(x \in X\) is a **decision vector**
- \(Z\): objective space, and \(z \in Z\) is a **objective vector**
- \(f: X \to Z\) is a function that assigns to each decision vector \(x\) an objective vector \(z = f(x)\)
- We can think of it \(z = (f_1, \ldots, f_n) = f(x_1, \ldots, x_m)\) while assuming \(Z = \mathbb{R}^n\)

- \(\preceq\) is a binary relation over \(Z\) that defines a **partial order** of the objective space (it also induces a preorder on the decision space)
Weak Dominance

When \( n = 1 \) (single objective function):

- Optimization problem: \((X, \mathbb{R}, f, \preceq)\)
- \( \preceq \) is our good old \( \leq \) on reals; there always exists a unique maximum

When \( n > 1 \) (multiple objective functions)

- Typically define \( \preceq \) as \( z^{(a)} \preceq z^{(b)} \equiv \forall i \in \{1 \ldots n\} z_i^{(a)} \leq z_i^{(b)} \)
- Known as weak Pareto dominance: \( z^{(b)} \) weakly dominates \( z^{(a)} \)
Optimization Problem

Goal:
Find solution $x^*$ that is mapped to a maximal element $z^* = f(x^*)$ in the set $f(X) = \{z \in Z \mid \exists x \in X : z = f(x)\}$

Think: $x$ is *program configuration*,
$z$ is pair (accuracy, speedup), and
$f$ computes (or records) accuracy and time of the execution.

- We can define the problem similar for searching minimal element (accuracy loss, run time)
- We can also make three dimensional tradeoff space accuracy, performance, energy, or even multidimensional
Our Optimization Problem

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

$$\text{maximize } (\text{Speedup}(X,i), \text{Accuracy}(X,i))$$

forall $i \in \text{InputSet}$

But these are most often competing objectives.

Consider turning into weighted single optimization problem ($w_{1,2}$ express preference):

$$\text{maximize } w_1 \times \text{Speedup}(X,i) + w_2 \times \text{Accuracy}(X,i)$$

forall $i \in \text{InputSet}$

To maintain accuracy guarantees rephrase: for every accuracy loss threshold $\delta$

$$\text{maximize } \text{Speedup}(X,i)$$

subject to $\text{AccuracyLoss}(X,i) \leq \delta$

forall $i \in \text{InputSet}$
Dominance

\( z^{(a)} \preceq z^{(b)} \) for objective vectors of size \( n \) is defined as

\[
\forall i \in \{1 \ldots n\} . \ z_i^{(a)} \leq_{\mathbb{R}} z_i^{(b)}
\]

It is also called weak Pareto dominance

A strong Pareto dominance \( z^{(a)} \prec z^{(b)} \) is defined as above, but cannot have any element being equal.

Read:

- \( z^{(a)} \preceq z^{(b)} \) we say that \( z^{(b)} \) weakly dominates \( z^{(a)} \)
- \( z^{(a)} \prec z^{(b)} \) we say that \( z^{(b)} \) dominates \( z^{(a)} \)

We can similarly define this relation for the cases when we want to maximize one but minimize another objective.
Dominance

We just learned about **Pareto Dominance** (and weak dominance)

**Incomparable points:** neither $z^a \preceq z^b$ nor $z^b \preceq z^a$

**Indifferent:** both points have the same value in all objectives

**Strict domination:** $z^a$ is better than $z^b$ in all objectives

**Utopia point (★):** result of individual objective optimizations
Pareto Set Approximations

In optimization we are interested in the entire Pareto-optimal set, not just individual solutions

- The set comprises the non-dominated objectives and decisions:* 
  \[ A = \{(z, x) \mid \exists x \in X \exists z \in Z \text{ s.t. } z = f(x) \text{ and } z \text{ in not dominated}\} \]

- We want to find mutually incomparable solutions
- Each such solution is a Pareto set approximation

We can extend the optimization problem: we want the best set of Pareto points (over other sets)

- Think: we want the best tradeoff curve across all that can be computed

---

*With a small abuse of notation, \( z_B \in B \) refers to \( (z_B, x_B) \in B \) for some \( x_B \) but the decision vector \( x_B \) is not necessary in this context; Alternatively, one could write \( (z_B, -) \in B \). We treat the case \( x_B \in B \) the same way.
Algorithm 2: Simple Cull minimisation algorithm

```
SimpleCull(C)
C_{min} := \emptyset;
while C \neq \emptyset do
  c := RemoveElement(C); dominated := false;
  foreach d \in C_{min} do
    if c \preceq d then C_{min} := C_{min} \setminus \{d\};
    else if d \preceq c then dominated := true; break;
  end
  if not dominated then C_{min} := C_{min} \cup \{c\};
end
return C_{min};
```

A Calculator for Pareto Points, Marc Geilenand and Twan Basten (2008)
Comparing Pareto Sets Approximations

Let A and B the sets of Pareto-optimal points (e.g., produced by different search algorithms or multiple runs of a randomized algorithm)

Is this enough? Typically no, we may need to define quality indicators to compare ‘incomparable’ sets

• There is no standard quality indicator, but needs to be selected based on context

Hypervolume Indicator: intuitively, a volume (in our case area) of dominated solutions covered by the Pareto set.

• Need to select a reference point (or points). In our case, (speedup,accuracy) pairs (1.0, 100%) and (1.0, max-acceptable-accuracy) are intuitive choices

• Can order the Pareto sets $I(A) > I(B) \implies A \triangleright B$ (i.e., A is better than B)

• For randomized search algorithms, can compute and compare expected indicators i.e., $\mathbb{E} I(A) > \mathbb{E} I(B) \implies A \triangleright B$
Pareto Fronts (aka Tradeoff curves)
Pareto Fronts (aka Tradeoff curves)
Pareto Fronts (aka Tradeoff curves)
Comparing Pareto Set Approximations

Let A and B the sets of Pareto-optimal points (e.g., produced by different search algorithms or multiple runs of a randomized algorithm)

We can define the relations for the sets:

- **A dominates B** \((B < A)\) iff every \(z_B \in B\) is dominated by some \(z_A \in A\)
- **Weak domination** \((B \preceq A)\) is defined similarly
- **A and B are indifferent:** A weakly dominates B and B weakly dominates A
- **A is better than B** \((B \prec A)\): every \(z_B \in B\) is weakly dominated by at least one \(z_A \in A\) and A and B are not indifferent
- **A and B are incomparable:** neither set weakly dominates the other
- Is this enough? Typically no, we may need to define quality indicators to compare ‘incomparable’ sets.
Note on Our Optimization

Since we execute the programs, the input distributions will impact the approximation sets

Alternatively, if we combine with static analysis, some of the tradeoffs will end up being conservatively set

The search algorithms (e.g., auto-tuners) will impact what solutions we find – especially if they are randomized

The distribution between the ‘training’ and ‘test’ inputs may change, impacting accuracy and performance
LET’S START WITH PRACTICE THEN
**Petabricks**

Language for algorithmic choice (expresses options to tune) and an autotuner (using genetic search)

Precusor to OpenTuner (popular autotuner: [http://opentuner.org](http://opentuner.org))

Hand-coded algorithmic compositions are commonplace. A typical example of such a composition can be found in the C++ Standard Template Library (STL) routine `std::sort`, which uses merge sort until the list is smaller than 15 elements and then switches to insertion sort. Our tests have shown that higher cutoffs (around 60-150) perform much better on current architectures. However, because the optimal cutoff is dependent on architecture, cost of the comparison routine, element size, and parallelism, no single hard-coded value will suffice.
Petabricks

Language for algorithmic choice (expresses options to tune) and an autotuner (using genetic search)

Precursor to OpenTuner (popular autotuner)

Classes of algorithms that can benefit from approximation:

- Polyalgorithms
- NP-Complete Algorithms
- Iterative Algorithms
- Signal Processing
The rules contained in the body of the transform define the various pathways to construct the Assignments data from the initial Points data.

```
transform kmeans
accuracy_metric kmeansaccuracy
accuracy_variable k
from Points[n,2] // Array of points (each column
  // stores x and y coordinates)
through Centroids[k,2]
to Assignments[n]
{
  ... (Rules 1 and 2 same as in Figure 1) ...

  // Rule 3:
  // The kmeans iterative algorithm
  to(Assignments a) from(Points p, Centroids c) {
    for_enough {
      change;
      AssignClusters(a, change, p, c, a);
      if (change==0) return; // Reached fixed point
      NewClusterLocations(c, p, a);
    }
  }
}
```

```
transform kmeansaccuracy
from Assignments[n], Points[n,2]
to Accuracy
{
  Accuracy from(Assignments a, Points p) {
    return sqrt(2*n/SumClusterDistanceSquared(a, p));
  }
}
```
Petabricks Autotuner

```plaintext
transform kmeans
accuracy_metric kmeansaccuracy
accuracy_variable k
from Points[n,2] // Array of points (each column
  // stores x and y coordinates)
through Centroids[k,2]
to Assignments[n]
{
  ... (Rules 1 and 2 same as in Figure 1) ...
  // Rule 3:
  // The kmeans iterative algorithm
to(Assignments a) from(Points p, Centroids c) {
    for_enough {
      int change;
      AssignClusters(a, change, p, c, a);
      if (change==0) return; // Reached fixed point
      NewClusterLocations(c, p, a);
    }
  }
}
transform kmeansaccuracy
from Assignments[n], Points[n,2]
to Accuracy
{
  Accuracy from(Assignments a, Points p){
    return sqrt(2*n/SumClusterDistanceSquared(a,p));
  }
}
```
Next Step

What if a language does not expose approximation choices?

Let a compiler find and expose some by modifying the program!
**SpeedPress**

- **Transforms** programs with perforation
- **Validates** new programs using testing

- Quality of Service Profiling (ICSE 2010)
- Managing Performance vs. Accuracy Trade-offs With Loop Perforation (FSE 2011)
Typical Inputs

Accuracy Specification

- **Quality Metric:** e.g. PSNR and bit rate
- **Quality Loss:** e.g. relative difference $<10\%$

**x264 Video Encoder Example**
Phases of Approximate Compiler:

*Find perforatable loops*

- **Identify Opportunity:** Run *performance* profiler
  - Identify time consuming loops

- **Sensitivity Testing:** Perforate *one loop* at a time
  - Filter out loops that do not satisfy accuracy requirement

- **Search for Optimal Knobs:** Perforate *multiple loops*
  - Find combinations of loops that maximize performance
  - Return a tradeoff curve of best solutions found
Validate Perforated Loops

Filter out loops that do not satisfy requirement

Criticality (Sensitivity) Testing: Ensure that the program with perforated loop does not:

- Crash or return error
- Runs slower than original (or not terminates)
- Causes other errors identified by dynamic analysis (e.g., latent memory errors)
- Produces unacceptable result (e.g., NaN, inf…)
- Produces inaccurate result (according to accuracy metric)
Criticality (Sensitivity) Testing:

*Filter out loops that do not satisfy requirement*

\[ \alpha(\cdot) = \begin{cases} \text{PSNR} & \text{bitrate} \\ q(\cdot, \cdot) \leq q_m \end{cases} \]
Perforating Individual Loops in x264

( Quality Loss < 0.1)
Perforating Individual Loops in x264

( Quality Loss < 0.1)
Perforating Individual Loops in \textit{x264}

( Quality Loss < 0.1)
Perforating Individual Loops in x264
(Quality Loss < 0.1)

# loops

- No Speedup
- Latent Errors
- Crash
Perforating Individual Loops in x264

( Quality Loss < 0.1)
Perforating Individual Loops in \textit{x264}

( Quality Loss < 0.1)

6 perforatable loops

- Green: Perforatable
- Orange: Low Accuracy
- Blue: No Speedup
- Red: Latent Errors
- Dark Red: Crash
Status

We found approximate computations and exposed individual knobs

Next, let us combine the knob values to utilize the approximation “budget”
Search Strategies and Algorithms

- Greedy
- Exhaustive
- Combined
- Hill-climbing
- Simulated annealing
- Genetic algorithm
- Reinforcement learning
- ...

- We had the comfort to do a bounded-exhaustive evaluation to explore the tradeoff space
Navigate Tradeoff Space

Quality loss

Speedup
Applications
From PARSEC Suite

**x264**  video encoder
**bodytrack**  human motion tracking
**swaptions**  financial analysis
**ferret**  image search
**canneal**  electronic circuit placement
**streamcluster**  point clustering
**blackscholes**  financial analysis
Inputs

Augmented or Replaced Existing Sets

- **x264** from Internet
- **bodytrack** augmented
- **swaptions** randomly generated
- **ferret** provided inputs
- **canneal** augmented (autogenerated)
- **streamcluster** from Internet
- **blackscholes** provided inputs
## Metrics

*Application Specific*

<table>
<thead>
<tr>
<th>Application</th>
<th>Metric</th>
</tr>
</thead>
<tbody>
<tr>
<td>x264</td>
<td>PSNR + Size</td>
</tr>
<tr>
<td>bodytrack</td>
<td>weighted relative difference</td>
</tr>
<tr>
<td>swaptions</td>
<td>relative difference</td>
</tr>
<tr>
<td>ferret</td>
<td>recall</td>
</tr>
<tr>
<td>canneal</td>
<td>relative difference</td>
</tr>
<tr>
<td>streamcluster</td>
<td>clustering metric</td>
</tr>
<tr>
<td>blackscholes</td>
<td>relative difference</td>
</tr>
</tbody>
</table>
Loop Perforation  
*(Quality Loss < 10%)*

<table>
<thead>
<tr>
<th>Application</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>x264</td>
<td>3.2x</td>
</tr>
<tr>
<td>bodytrack</td>
<td>6.9x</td>
</tr>
<tr>
<td>swaptions</td>
<td>5.0x</td>
</tr>
<tr>
<td>ferret</td>
<td>1.1x</td>
</tr>
<tr>
<td>canneal</td>
<td>1.2x</td>
</tr>
<tr>
<td>streamcluster</td>
<td>1.2x</td>
</tr>
</tbody>
</table>
Loop Perforation
(Quality Loss < 10%)

- x264 3.2x motion estimation
- bodytrack 6.9x particle filtering
- swaptions 5.0x MC simulation
- ferret 1.1x image similarity
- canneal 1.2x simulated annealing
- streamcluster 1.2x cluster center search
Loop Perforation
(Quality Loss < 10%)

Tasks of most perforated loops:
• Distance metrics
• Search-space enumeration
• Iterative improvement
• Redundant executions

x264
bodytrack
swaptions
ferret
canneal
streamcluster
Main Observations

- **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)
Approximate Program Analysis = Accuracy + Safety
Accuracy and Guarantees

Logic-Based *(worst-case)*
“for all inputs…”

Probabilistic *(worst-case or average-case)*
“for all inputs, with probability at least p…”
“for inputs distributed as…”

Statistical *(average-case)*
“for inputs distributed as… with confidence c”
“for tested inputs… with confidence c”

Empirical *(typical-case)*
“for typical inputs…”
Goals of Runtime Adaptation

Accuracy (Green)

Time or Energy
(Loop perforation)
Green : Framework for Controlled Approximations (PLDI’10) *

End-to-end framework for controlled application on approximations
  • Loop and function approximations

Relatively easy for programmers to use

Hooks for expert programmers and custom policies

Online mechanism to reactively adapt approximation policy to meet QoS

Adopted from slides by Radha Venkategiri
Recalibration

**Concern:** Overhead for running non-approximate

**Address:** Run infrequently, restructure the code

![Figure 3. An end-to-end example of applying loop approximation to the Pi estimation program.](image-url)
Recalibration

```cpp
QoS_ReCalibrate(QoS_loss, QoS_SLA) {
    // n_m: number of monitored queries
    // n_l: number of low QoS queries in monitored queries
    if (n_m==0) {
        // Set Sample_QoS to 1 to trigger QoS_ReCalibrate
        // for the next 100 consecutive queries
        Saved_Sample_QoS=Sample_QoS;
        Sample_QoS=1;
    }
    n_m++;
    if (QoS_loss !=0)
        n_l++;
    if (n_m==100) {
        QoS_loss=n_l/n_m;
        if(QoS_loss>QoS_SLA) {
            // low QoS case
            increase_accuracy();
        } else if (QoS_loss < 0.9*QoS_SLA) {
            // high QoS case
            decrease_accuracy();
        } else {
            // no change
            Sample_QoS=Saved_Sample_QoS;
        }
    }
}
```

**Figure 9.** Customized QoS_ReCalibrate for Bing Search.
Runtime Adaptation for Accuracy

Key concerns:

• **Reexecuting infrequently to reduce the overhead**
  checking every result is expensive, rely on spatial and temporal locality

• **The computation needs to be amenable for re-execution:**
  think no side effects or crashes due to approximation
At regular time intervals we gradually reduce the frequency of the SoC.

When you notice a disruption, read the value from the tradeoff curve that would negate the disruption.
Runtime tuning helps maintain responsiveness in face of frequency changes.
Runtime Approximation for Time/Energy

Runtime tuning helps maintain responsiveness in face of frequency changes
Runtime Approximation for Time/Energy

Runtime tuning helps maintain responsiveness in face of frequency changes
Runtime Approximation for Time/Energy

Runtime tuning helps maintain responsiveness in face of frequency changes
Runtime Approximation for Time/Energy

Runtime tuning helps maintain responsiveness in face of frequency changes
Runtime Approximation for Time/Energy

Runtime tuning helps maintain responsiveness in face of frequency changes
Runtime tuning helps maintain responsiveness in face of frequency changes
Runtime Approximation for Time/Energy

Runtime tuning helps maintain responsiveness in face of frequency changes.
What if you don’t have the exact point?

Solution 1: Select more conservative, suffer some performance drop (point A)

Solution 2: Select more aggressive, lose some more accuracy and make program even faster (point B)

Solution 3: We can use randomization

- Choose point A with probability $p$ and
- Choose point B with probability $1-p$

Why would this work over a long sequence of runs?
Analysis-driven Compiler

SAS '11, POPL '12, OOPSLA'13, OOPSLA '14, OOPSLA'19, CGO '20

- **Statically analyze** computation’s accuracy
- **Transform** computation by solving a mathematical optimization problem

Optimized Computation
Recall: Our Optimization Problem

Select Program Configuration $X \in Configs$ to

$$\text{maximize } (\text{Speedup}(X, i), \text{Accuracy}(X, i))$$

forall $i \in InputSet$

But these are most often competing objectives.

Consider turning into weighted single optimization problem ($w_{1,2}$ express preference):

$$\text{maximize } w_1 \times \text{Speedup}(X, i) + w_2 \times \text{Accuracy}(X, i)$$

forall $i \in InputSet$

To maintain accuracy guarantees rephrase: for every accuracy loss threshold $\delta$

$$\text{maximize } \text{Speedup}(X, i)$$

subject to $\text{AccuracyLoss}(X, i) \leq \delta$

forall $i \in InputSet$
Let’s Extend It a Bit:

\[
\begin{align*}
\text{maximize} & \quad Speedup(X, i) \\
\text{subject to} & \quad AccuracyLoss(X, i) \leq \delta \\
& \quad SafetyPredicate(X, i) = \text{True} \\
\text{forall} & \quad i \in InputSet
\end{align*}
\]
Let’s Extend It a Bit:

\[
\text{maximize } \text{Speedup}(X, i) \\
\text{subject to } \text{AccuracyLoss}(X, i) \leq \delta \\
\text{SafetyPredicate}(X) = \text{True} \\
\text{forall } i \in \text{InputSet}
\]
Approximate Program Safety:
Information-flow Type Systems
Relational Logic Reasoning
EnerJ Type System

Idea:
Isolate code and data that must be precise from those that can be approximated

Sampson, Dietl, Fortuna, Gnanapragasam, Ceze, Grossman
EnerJ: Approximate Data Types for Safe and General Low-Power Computation (PLDI 2011)
Approximate Hardware Model from EnerJ

Recall – hardware approximations:
- Soft errors
- Timing errors
- Voltage variations
- Aging, Refresh rates, …

All can be modeled as wrong bits (permanent or transient)
EnerJ Type System

Idea:
Isolate code and data that must be precise from those that can be approximated

```
@Approx int a = approximate_code();
int p;
p = a;  <------- not ok
```
@Approximable class FloatSet {
    @Context float[] nums = ...;  // can be approximate or exact
    float mean() {
        float total = 0.0f;
        for (int i = 0; i < nums.length; ++i) total += nums[i];
        return total / nums.length;
    }

    @Approx float mean APPROX() {
        @Approx float total = 0.0f;
        for (int i = 0; i < nums.length; i += 2) total += nums[i];
        return 2 * total / nums.length;
    }
}

Use:
@approx FloatSet afs
@precise FloatSet fs
afs.mean() // call approx
fs.mean() // call precise
Formulate the Rules

Types: @approx, @precise, @context (the precision of the type depends on the precision of the enclosing object), @top (either of approx. or precise), @lost (information about type lost)

Subtyping. Subtyping is defined using an ordering of the precision qualifiers and subclassing.

The following rules define the ordering of precision qualifiers:

\[ q <: q' \] ordering of precision qualifiers

\[
\begin{align*}
q \neq \text{top} & \quad q <:q \text{ lost} \\
q <:q \text{ top} & \quad q <:q q
\end{align*}
\]

Recall that top qualifies the common supertype of precise and approx types. Every qualifier other than top is below lost; every qualifier is below top; and the relation is reflexive. Note that the precise and approx qualifiers are not related.
EnerJ Type System

Idea:
Isolate code and data that **must be precise** from those that **can be approximated**

```java
@Approx FloatSet afs;
@Precise FloatSet fs;

@Approx int a = fs.mean(); // call approx
@Precise int p = fs.mean(); // call precise

print (a);
print (p);
```
EnerJ Type System

Idea:
Isolate code and data that must be precise from those that can be approximated

@Approx int a = approximate_code();
int p;
if (a > 3) { p = 1; } else { p = 2; }

Control flow dependency (implicit flow)
Typing Rule

\[ s\Gamma \vdash e_0 : \text{precise } P \quad s\Gamma \vdash e_1 : T \quad s\Gamma \vdash e_2 : T \]

\[ s\Gamma \vdash \text{if}(e_0) \{ e_1 \} \text{ else } \{ e_2 \} : T \]

Approximate data cannot change the control flow
EnerJ Type System

Idea:
Isolate code and data that must be precise from those that can be approximated

```
@Approx int a = approximate_code();
int p;
p = endorse(a);  <------- ok
```

Like “(cast_type) a” in Java
EnerJ Type System

Consequence:
Then the approximate parts may be optimized automatically, but the developer needs to ensure the endorsed values are valid.

```java
@Approx int a = approximate_code();
int p;
p = endorse(a);  <-------- ok
if ( isValid(p) ) { ... } else { errorHandle(a) }
```
EnerJ Type System

Consequence:
Then the approximate parts may be optimized automatically, but the developer needs to ensure the endorsed values are valid.

```java
@Approx int a = approximate_code();
int p;
try {
    p = endorse(a);  // -------- ok
check ( isValid(p) ) { ... } // do the computation
recover { errorHandle(a) }
```
EnerJ Type System

Motivation:

Security information flow type systems – prevent the program from leaking information about private variables into public variables.

See: https://en.wikipedia.org/wiki/Information_flow_(information_theory)#Information_flow_control

Noninterference [Goguen and Meseguer 1982]:

“one group of users, using a certain set of commands is noninterfering with another group of users if the first group does with those commands can no effect on what the second group of users can see.”
General Verification?

For safety properties or worst-case accuracy properties

Two (not so simple steps):
1. Verify the original program wrt the property
2. Verify the approximate program wrt the property

But: how often can you do the step 1?
Relative Safety

If the original program satisfies all assertions, then the relaxed program satisfies all assertions.
Relative Safety vs. Just Safety

Established through any means: verification, testing, code review

If the original program satisfies all assertions, then the relaxed program satisfies all assertions.

Any inconsistent behavior must be in the original program!
How does it work?

1. Define operational and axiomatic semantics of a product program (it has the behaviors of the original and approximated programs)

2. Specify the approximation as a perturbation of the variables

3. Capture when the flow of control of the approximate program diverges from the original program

4. Capture when the flow of control of the approximate program converges back with the original program

5. The predicate to verify is on the variables that are from both programs
Relational Safety Verification

\[
\text{for } (i=0; i < m; i++) \\
\text{sum} = \text{sum} + x[i] \\
\}
\]
\[
\text{avg} = \frac{\text{sum}}{m}
\]

\[
i < 2 \times m/3 \\
i < m/2
\]
relax (m) st (0 < m <= old(m))

for (i=0; i < m; i++) {
    sum = sum + x[i]
}

avg = sum / m
Relational Safety Verification

\[
\text{relax (m) st (} 0 < m \leq \text{old}(m) \text{)}
\]

\[
\text{for (i=0; i < m; i++) { }
\quad \text{sum = sum + x[i]}
\]
\[
}\text{avg = sum / m}
\]

Transformed execution accesses only (a subset of) memory locations that the original execution would have accessed
Relational Safety Verification

relax (m) st (0 < m <= old(m))

for (i=0; i < m; i++) {
    sum = sum + x[i]
}

avg = sum / m

The difference between the variable in the original and approximate runs is at most $\delta$

$|\text{sum}(o) - \text{sum}(r)| \leq \delta$
Control-Flow Divergence

// when b is possibly approximate
if (b) {
    x = y - d
} else {
    x = y + d
}

relate ( |x<o> - x<r>| <= ?? )

Need to consider all four options: (1) original then – relaxed then; (2) original then – relaxed else (3) original else – relaxed then (3) original else – relaxed else
Control-Flow Divergence

// when b is possibly approximate
if (b) {
    x = y - d
    x<o> = y - d
    x<r> = y - d
}
else {
    x = y + d
    x<o> = y + d
    x<r> = y + d

    x<o> = y + d
    x<r> = y - d
}

relate ( |x<o> - x<r>| <= ?? )

Need to consider all four options: (1) original then – relaxed then; (2) original then – relaxed else (3) original else – relaxed then (3) original else – relaxed else
Control-Flow Divergence

// when b is possibly approximate
if (b) {
    x = y - d  
} else {
    x = y + d
}

relate ( |x<o> - x<r>| <= 2*d )

Need to consider all four options: (1) original then – relaxed then; (2) original then – relaxed else (3) original else – relaxed then (3) original else – relaxed else
Recall: Accuracy and Guarantees

Logic-Based (*worst-case*)
“for all inputs…”

Probabilistic (*worst-case or average-case*)
“for all inputs, with probability at least p…”
“for inputs distributed as…”

Statistical (*average-case*)
“for inputs distributed as… with confidence c”
“for tested inputs… with confidence c”

Empirical (*typical-case*)
“for typical inputs…”
Computation difference

\[ D = \text{res}_{\text{original}} - \text{res}_{\text{transformed}} \]
Worst-Case Analysis

\[ D = \text{res}_{\text{original}} - \text{res}_{\text{transformed}} \]
Alternative: Probabilistic Analysis

\[ D = \text{res}_{\text{original}} - \text{res}_{\text{transformed}} \]
Analysis-Based Optimizations

**Accuracy Specification**

**Reliability**  Function computes result correctly with probability $> 0.99$

**Absolute Error**  Absolute error of function’s result $< 2.0$

**Reliability and Absolute Error**  Absolute error of function’s result $< 2.0$ with probability $> 0.99$

$$\text{int}\{\Delta f \leq 2; 0.99 \ast R(\Delta x = 0, \Delta y = 0)\} \ f(\text{int} \ x, \text{int} \ y);$$
Reliability Specification

The function computes result correctly with probability at least 0.99.
Reliability Specification

Probability that the parameters have correct values before function starts executing *(facilitates function composition)*
Reliability Specification

- Reliability factor: $R(\Delta v_1 \leq d_1, \ldots, \Delta v_n \leq d_n)$

$$\Delta v \equiv |v_{exact} - v_{approx}|$$

Numerical bound
Function Optimization Problem

Find Function Configuration $q$:

\[
\max \text{ EnergySavings} (q) \\
\text{s.t.} \quad \text{Reliability} (q) \geq \text{ReliabilityBound} \\
\quad \text{AbsoluteError} (q) \leq \text{ErrorBound}
\]
Image Scaling: Interpolation Function

\[ f \left( \begin{array}{cccc}
& & & \\
& & & \\
& & & \\
& & & \\
\end{array} \right) = \]

[Image of a mandrill face before and after scaling]
Interpolation Function

```c
int interpolation(int dst_x, int dst_y, int src[][[]])
{
    int x = src_location_x(dst_x, src),
    y = src_location_y(dst_y, src);

    int up    = src[y - 1][x],
    down    = src[y + 1][x],
    left    = src[y][x - 1],
    right   = src[y][x + 1];

    int val = up + down + left + right;

    return 0.25 * val;
}
```
Approximate Hardware Model from EnerJ

Recall – hardware approximations:
• Soft errors
• Timing errors
• Voltage variations
• Aging, Refresh rates, ...
All can be modeled as wrong bits (permanent or transient)
Run Function on Approximate Hardware

```c
int interpolation(int dst_x, int dst_y, int src[][[]])
{
    int x = src_location_x(dst_x, src),
        y = src_location_y(dst_y, src);

    int up   = src[y - .1][x],
        down = src[y + .1][x],
        left = src[y][x - .1],
        right = src[y][x + .1];

    int val = up + down + left + right;

    return 0.25 * val;
}
```
int interpolation(int dst_x, int dst_y, int src[][])
{
    int x = src_location_x(dst_x, src),
        y = src_location_y(dst_y, src);

    int up    = src[y -. 1][x],
        down  = src[y +. 1][x],
        left  = src[y][x -. 1],
        right = src[y][x +. 1];

    int val = up +. down +. left +. right;

    return 0.25 * . val;
}
Function Configuration

Binary vector \( q = (q_1, q_2, \ldots, q_n) \)

Variable Declarations:
- \( q_i \) - if 1, variable is stored in approximate memory
  if 0, variable is stored in exact memory

Arithmetic Operations:
- \( q_i \) - if 1, the operation is approximate,
  if 0, the operation is exact
Function Configuration

```c
int interpolation(int dst_x, int dst_y, int src[][[]])
{
    int x = src_location_x(dst_x, src);
    int y = src_location_y(dst_y, src);

    int up    = src[y - 1][x];
    int down  = src[y + 1][x];
    int left  = src[y][x - 1];
    int right = src[y][x + 1];

    int val = up + down + left + right;

    return 0.25 * val;
}
```
Function Configuration

```c
int interpolation(int q_{dstx} dst_x, int q_{dsty} dst_y, int q_{src} src[][]) {
    int q_x x = src_location_x(dst_x, src);
    int q_y y = src_location_y(dst_y, src);

    int q_{up} up = src[y - 1][x];
    int q_{down} down = src[y + 1][x];
    int q_{left} left = src[y][x - 1];
    int q_{right} right = src[y][x + 1];

    int q_{val} val = up + down + left + right;
    return 0.25 * val;
}
```
Function Configuration

Each assignment of vector $q$ denotes a different approximate function

```c
int interpolation(int $q_{dstx}$ dst_x, int $q_{dsty}$ dst_y, int $q_{src}$ src[][[]])
{
    int $q_x$ x = src_location_x(dst_x, src);
    int $q_y$ y = src_location_y(dst_y, src);

    int $q_{up}$ up = src[y - $q_7$ 1][x];
    int $q_{down}$ down = src[y + $q_6$ 1][x];
    int $q_{left}$ left = src[y][x - $q_5$ 1];
    int $q_{right}$ right = src[y][x + $q_4$ 1];

    int $q_{val}$ val = up + $q_1$ down + $q_2$ left + $q_3$ right;

    return 0.25 * $q_0$ val;
}
```
Reliability Analysis

Motivation

• Efficiently represent reliability of all approximate function versions

• Construct constraints to separate those function versions that satisfy specification
Reliability Analysis

Approximate hardware specification:
- Reliability of arithmetic operations: $r_{op} \in (0, 1]$
- Reliability of memory reads and writes: $r_{rd}, r_{wr} \in (0, 1]$

\[
\text{operator} (\ast) = 0.9999; \\
\text{memory} \text{ approx} \{\text{rd} = 0.99998, \text{wr} = 0.99999\};
\]

Analysis:
- Sound static analysis, operates backward
- Constructs symbolic expressions that characterize reliability of kernel’s traces
Reliability Analysis

**Statement**

```python
return val * 0.25;
```

**Exact Statement**

`val and * exact`

1. Read val
2. Multiply
3. Return result

**Approximate Statement**

`val and * approximate`

1. `r_{rd}`
2. `1 - r_{rd}`
3. `r_{times}`
4. `1 - r_{times}`
Reliability Analysis

**Statement**

```
return val * 0.25;
```

**Exact Statement**

```
val and *
exact
```

**Approximate Statements**

```
val and *
approximate
```

### Statement reliability

1.0

```
r_{rd} \cdot r_{times}
```

```
r_{rd}
```

```
r_{times}
```
Reliability Analysis

Statement

Return statement:

\[
\text{return } \text{val} * 0.25; \]

Encode approximation choice:

• Variable declaration: \( \text{int} \ \text{q}\_{\text{val}} \ \text{val}; \)

• Multiplication: \( \text{val} * q* 0.25; \)
Reliability Analysis

Statement

\[
\text{return \; val \times 0.25;}
\]

Reliability Expression

\[
(r_{rd})^{q_{val}} \cdot (r_{times})^{q_{*}} \cdot R(\Delta val = 0)
\]

Encode approximation choice:

- Variable declaration: \( \text{int } q_{val} \text{ val;} \)
- Multiplication: \( \text{val } \times q_{*} \; 0.25; \)
Reliability Analysis

Statement

\[ \text{return \ val \ *} \ 0.25; \]

Reliability Expression

\[ (r_{rd})^{q_{val}} \cdot (r_{times})^{q_\star} \cdot R(\Delta \text{val} = 0) \]

Reliability of reading \text{val} from either exact or approximate memory:

\[ (r_{rd})^0 = 1.0 \]
\[ (r_{rd})^1 = r_{rd} \]
Reliability Analysis

**Statement**

```python
return val * 0.25;
```

**Reliability Expression**

\[
(r_{rd})^{q_{val}} \cdot (r_{times})^{q^*} \cdot R(\Delta val = 0)
\]

Reliability of either exact or approximate multiplication
Reliability Analysis

Statement:
```
return val * 0.25;
```

Reliability Expression:
$$(r_{rd})^{q_{val}} \cdot (r_{times})^{q_{*}} \cdot R(\Delta val = 0)$$

Probability that previous statements computed `val` correctly.
```c
int interpolation(int q_{dstx}, dst_x, int q_{dsty}, dst_y, int q_{src} src[][]) {
    return val * q_{*} 0.25;
}
```

**Interpolation Function**

\[(r_{rd})^{q_{val}} \cdot (r_{times})^{q_{*}} \cdot R(\Delta val = 0)\]
Interpolation Function

```c
int interpolation(int q_dstx, dst_x, int q_dsty, dst_y, int q_src[src][[]])
{
  int q_val = up + q_1 down + q_2 left + q_3 right;
  return val * q_0.25;
}
```

\[(r_{rd})^{q_{val}} \cdot (r_{times})^{q_0} \cdot (r_{plus})^{q_1+q_2+q_3} \cdot (r_{rd})^{q_{up}+q_{down}+q_{left}+q_{right}} \cdot R(\Delta_{up} = 0, \Delta_{down} = 0, \Delta_{left} = 0, \Delta_{right} = 0)\]
Reliability Expression

Function’s Reliability Expression:

\[ r_1^{q_1} \cdot r_2^{q_2} \cdot \ldots \cdot r_n^{q_n} \cdot R \]

- Probability operations executed reliably (for all approximate versions of the function)
- Probability parameters have correct values at function start

\[ \Delta_{dst_x} = 0, \quad \Delta_{dst_y} = 0, \quad \Delta_{src} = 0 \]
Reliability Constraint

Relates developer’s specification and analysis result:

\[ r_{\text{spec}} \cdot R(P_{\text{spec}}) \leq r_1^{q_1} \cdot r_2^{q_2} \cdots r_n^{q_n} \cdot R(P_{\text{param}}) \]
Reliability Constraint

\[ r_{spec} \leq r_1^{q_1} \cdot r_2^{q_2} \cdot \ldots \cdot r_n^{q_n} \]

and

\[ R(P_{spec}) \leq R(P_{param}) \]

Can Immediately Solve
Reliability Constraint

\[ r_{spec} \leq r_1^{q_1} \cdot r_2^{q_2} \cdot \ldots \cdot r_n^{q_n} \]

and

\[ R(P_{spec}) \leq R(P_{param}) \]

\[ \Delta_{dst_x} = 0, \quad \Delta_{dst_y} = 0, \quad \Delta_{src} = 0 \]

\[ P_{spec} \Rightarrow P_{param} \]

\[ \Delta_{dst_x} = 0, \quad \Delta_{dst_y} = 0, \quad \Delta_{src} = 0 \]
Reliability Constraint

\[ r_{spec} \leq r_1^{q_1} \cdot r_2^{q_2} \cdot \ldots \cdot r_n^{q_n} \]

Denotes approximate function versions that satisfy the developer’s specification.
Reliability Constraint

for the optimization problem

$$\log(r_{spec}) \leq q_1 \cdot \log(r_1) + q_2 \cdot \log(r_2) + \cdots + q_n \cdot \log(r_n)$$

Denotes approximate function versions that satisfy the developer’s specification
## Reliability and Control Flow

<table>
<thead>
<tr>
<th>Conditionals</th>
<th>Constraints for each program path</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Analysis removes redundant constraints</td>
</tr>
<tr>
<td></td>
<td>(most constraints can be removed - OOPSLA ’13)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bounded Loops</th>
<th>Statically known loop bound</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Analysis unrolls loop</td>
</tr>
</tbody>
</table>

| Optimization Granularity | Optimize blocks of code instead of individual instructions |
Function Optimization Problem

Find Function Configuration $q$: ✓

\[
\max \text{ EnergySavings}(q) \\
\text{Reliability}(q) \geq \text{ReliabilityBound} \\
\text{AbsoluteError}(q) \leq \text{ErrorBound}
\]
Absolute Error Analysis

Reduced-precision floating-point instructions:
• *Almost always* incorrect, but error is bounded
• Hardware specification: number of significant mantissa bits

Analysis:
• Bounds worst-case numerical deviation
• Embeds accuracy predicate in reliability factor:

\[ r_1^{q_1} \cdot \ldots \cdot r_n^{q_n} \cdot R(\Delta x = 0, \Delta y = 0) \]
Absolute Error Analysis

Reduced-precision floating-point instructions:
- **Almost always** incorrect, but error is bounded
- Hardware specification: number of significant mantissa bits

Analysis:
- Bounds worst-case numerical deviation
- Embeds accuracy predicate in reliability factor:

\[
r_1^{q_1} \cdot \ldots \cdot r_n^{q_n} \cdot R(\Delta x + 2 \cdot \Delta y + q_1 \cdot \xi_{x,y} < d)
\]

Linear function of \(q_1, \ldots, q_n\)
Recall Error Propagation

\[ |f_1(X + \varepsilon_0) + \varepsilon_1 - f_1(X)| \leq K_{f_1} \cdot \varepsilon_0 + \varepsilon_1 \]

\[ |f_2(f_1(X + \varepsilon_0) + \varepsilon_1) + \varepsilon_2 - f_2(f_1(X))| \leq K_{f_2} \cdot K_{f_1} \cdot \varepsilon_0 + K_{f_2} \cdot \varepsilon_1 + \varepsilon_2 \]

\[ K_f = \max_{x \in \text{Inputs}} \left| \frac{df}{dx} \right| \]
Error Propagation for Some Common Functions

\[ K_f = \max_{x \in \text{Inputs}} \left| \frac{df}{dx} \right| \]

\[ K_{fi} = \max_{x \in \text{Inputs}} \left| \frac{\partial f(x_1 \ldots x_n)}{\partial x_i} \right| \]

- \( f(x_1, x_2) \) 
  - Error
- \( x \cdot \text{const} \)
  - \( \Delta x \cdot \text{const} \)
- \( x + y \)
  - \( \Delta x + \Delta y \)
- \( x \cdot y \)
  - \( \Delta x \cdot \max(|y + \Delta y|) + \Delta y \cdot \max(|x + \Delta x|) \)
Interpolation Function

```c
int interpolation(int q_dstx, dst_x, int q_dsty, dst_y, int q_src[][[]])
{
    return val * q * 0.25;
}
```

\[
(r_{rd})^{q_{val}} \cdot (r_{times})^{q_{*}} \cdot R(0.25 \cdot \Delta val + q_{*} \cdot e_{*} \leq E)
\]

return val * q * 0.25;
**Interpolation Function**

```c
int interpolation(int q_{dstx} dst_x, int q_{dsty} dst_y, int q_{src} src[][])
{
    int q_{val} val = up + q_{1} down + q_{2} left + q_{3} right;

    (r_{rd})^{q_{val}} \cdot (r_{times})^{q_{\ast}} \cdot (r_{plus})^{q_{1}+q_{2}+q_{3}} \cdot (r_{rd})^{q_{up}+q_{down}+q_{left}+q_{right}}
    \cdot R\left(0.25 \cdot (\Delta up + \Delta down + \Delta left + \Delta right) + 0.25 \cdot (q_{1} \cdot e_{1} + q_{2} \cdot e_{2} + q_{3} \cdot e_{3}) + q_{\ast} \cdot e_{\ast} \leq E\right)

    return val \cdot q_{\ast} \cdot 0.25;
}
```
Function Optimization Problem

Find Function Configuration $q$:

$$\max \text{EnergySavings} \ (q)$$

Reliability $(q) \geq \text{ReliabilityBound}$

AbsoluteError $(q) \leq \text{ErrorBound}$
Energy Savings Analysis

Profile information:
• Collects traces from running representative inputs

Analysis:
• Estimates savings for instructions and variables from traces

\[
\begin{align*}
\text{instruction} & : \quad q_\ell \cdot Count_\ell \cdot Saving_{ALU} \\
\text{variable} & : \quad q_m \cdot Size_m \cdot Saving_{MEM}
\end{align*}
\]
Energy Savings Analysis

Profile information:
• Collects traces from running representative inputs

Analysis:
• Estimates savings for instructions and variables from traces

Approximate hardware specification:
• Relative savings for operations and memories
• Percentage of system energy that ALU and memory consume

\[
\begin{align*}
&c_{\text{ALU}} \sum_{\ell \in \text{Instr}} q_\ell \cdot \text{Count}_\ell \cdot \text{Saving}_{\text{ALU}} + c_{\text{MEM}} \sum_{m \in \text{Var}} q_m \cdot \text{Size}_m \cdot \text{Saving}_{\text{MEM}}
\end{align*}
\]
Function Optimization Problem

Find Function Configuration $q$:

$$\max \text{ EnergySavings} (q)$$

Reliability $(q) \geq \text{ReliabilityBound}$

AbsoluteError $(q) \leq \text{ErrorBound}$
Find Function Configuration $q$:

\[ \text{max} \quad \text{EnergySavings}(q) \]
\[ \text{Reliability}(q) \geq \text{ReliabilityBound} \]
\[ \text{AbsoluteError}(q) \leq \text{ErrorBound} \]

Solve using off-the-shelf solvers (we use Gurobi)
Evaluation

Benchmarks With Approximated Functions:

- **Scale**: image scaling
- **DCT**: discrete cosine transform
- **IDCT**: inverse discrete cosine transform
- **Blackscholes**: financial option price calculation
- **SOR**: successive over-relaxation kernel

Approximate Hardware Specifications:

- 5 specifications of ALU, caches, and memories from the literature [EnerJ – PLDI ’11]
## Complexity of Optimization Problem

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Function LOC</th>
<th>Search Space Size</th>
<th>Reliability Constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scale</td>
<td>88</td>
<td>$2^{74}$</td>
<td>4</td>
</tr>
<tr>
<td>DCT</td>
<td>62</td>
<td>$2^{35}$</td>
<td>1</td>
</tr>
<tr>
<td>IDCT</td>
<td>93</td>
<td>$2^{53}$</td>
<td>1</td>
</tr>
<tr>
<td>Blackscholes</td>
<td>143</td>
<td>$2^{80}$</td>
<td>2</td>
</tr>
<tr>
<td>SOR</td>
<td>23</td>
<td>$2^{10}$</td>
<td>1</td>
</tr>
</tbody>
</table>

Solver finds optimal solutions in less than a second
Energy/Accuracy Tradeoffs

Optimizer computes estimated system savings
Maximum estimated savings for hardware specifications:

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Reliability Degradation</th>
<th>System-Level Energy Savings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scale</td>
<td>0.995</td>
<td>19.4%</td>
</tr>
<tr>
<td>DCT</td>
<td>0.99992</td>
<td>8.7%</td>
</tr>
<tr>
<td>IDCT</td>
<td>0.992</td>
<td>13.4%</td>
</tr>
<tr>
<td>Blackscholes</td>
<td>0.999</td>
<td>9.8%</td>
</tr>
<tr>
<td>SOR</td>
<td>0.995</td>
<td>19.8%</td>
</tr>
</tbody>
</table>
Pros:
- Can explore the space induced by much finer grained transformations (e.g., numerical precision)
- The results are valid for all inputs within range
- New analyses were developed in the meantime

Cons:
- Static analysis is much more conservative than testing
- The set of supported programs is limited
Analysis: Middle Road

What if we know the distribution of the inputs?
CASE 1: Sum Computation

• Original sum computation
  
  \[
  s = 0; \\
  \text{for (i = 0; i < n; i++) } s = s + f(i);
  \]

• Perforated, extrapolated sum computation
  
  \[
  s = 0; \\
  \text{for (i = 0; i < n; i += 2) } s = s + f(i); \\
  s = s * 2;
  \]
Step 1: Represent Result Difference

• Original sum computation
  
  \[
  s = 0; \\
  \text{for } (i = 0; i < n; i++) \quad s = s + f(i);
  \]

• Perforated, extrapolated sum computation
  
  \[
  s = 0; \\
  \text{for } (i = 0; i < n; i += 2) \quad s = s + f(i); \\
  s = s \times 2;
  \]

• Perforation noise: \( D = s_{\text{original}} - s_{\text{perforated}} \)
Step 2: Probabilistic Modeling

• Original sum computation
  \[
  s = 0; \\
  \text{for } (i = 0; i < n; i++) \quad s = s + f(i); \\
  \]

• Perforated, extrapolated sum computation
  \[
  s = 0; \\
  \text{for } (i = 0; i < n; i += 2) \quad s = s + f(i); \\
  s = s \times 2; \\
  \]

• Perforation noise: \[ D = s_{\text{original}} - s_{\text{perforated}} \]
Step 2: Probabilistic Modeling

• Original sum computation
  \[ s = 0; \]
  \[ \text{for } (i = 0; i < n; i++) \quad s = s + X_i; \]

• Perforated, extrapolated sum computation
  \[ s = 0; \]
  \[ \text{for } (i = 0; i < n; i += 2) \quad s = s + X_i; \]
  \[ s = s \times 2; \]

• Perforation noise: \[ D = s_{\text{original}} - s_{\text{perforated}} \]
Analysis: Input/Output Relation

Perforation noise:
\[ D = s_{\text{original}} - s_{\text{perforated}} \]
Analysis: Input/Output Relation

Perforation noise:

\[ D = s_{\text{original}} - s_{\text{perforated}} \]

\[ = X_0 + X_1 + X_2 + X_3 + X_4 + X_5 + X_6 + X_7 + \ldots \]

\[ - 2 \cdot (X_0 + X_2 + X_4 + X_6 + \ldots) \]
Analysis: Input/Output Relation

Perforation noise*:

\[ D = s_{\text{original}} - s_{\text{perforated}} \]

\[ = X_0 + X_1 + X_2 + X_3 + X_4 + X_5 + X_6 + X_7 + \ldots \]

\[ - X_0 - X_0 - X_2 - X_2 - X_4 - X_4 - X_6 - X_6 - \ldots \]

*Assuming for simplicity that the number of elements is even
Analysis: Input/Output Relation

Perforation noise*:

\[ D = s_{\text{original}} - s_{\text{perforated}} \]

\[ = X_0 + X_1 + X_2 + X_3 + X_4 + X_5 + X_6 + X_7 + \ldots \]

\[ - X_0 - X_0 - X_2 - X_2 - X_4 - X_4 - X_6 - X_6 - \ldots \]

\[ = \sum_{0\leq i < \frac{n}{2}} (X_{2i+1} - X_{2i}) \]

* Assuming for simplicity that the number of elements is even
Analysis Results

Perforation noise:
\[ D = \varphi(X_0, X_2, \ldots, X_{n-1}) \]
Analysis Results

Perforation noise: $D = \phi(X_0, X_2, \ldots, X_{n-1})$

Location: Mean $E(D) = \mu$
Analysis Results

Perforation noise:
\[ D = \varphi(X_0, X_2, \ldots, X_{n-1}) \]

Location: Mean
\[ E(D) = \mu \]

Spread: Variance
\[ \text{Var}(D) = \sigma^2 \]
Analysis Results

Perforation noise:
\[ D = \varphi(X_0, X_2, \ldots, X_{n-1}) \]

Location: Mean
\[ E(D) = \mu \]

Spread: Variance
\[ \text{Var}(D) = \sigma^2 \]

Bound: Distribution tail
\[ \Pr[|D| > \delta] < \varepsilon \]
Next Time

Probabilistic programming:
Democratizing probabilistic inference