Probabilistically Accurate Program Transformations

Sasa Misailovic, Daniel M. Roy, and Martin C. Rinard

MIT CSAIL
{misailo, droy, rinard}@csail.mit.edu

Abstract. The standard approach to program transformation involves the use of discrete logical reasoning to prove that the transformation does not change the observable semantics of the program. We propose a new approach that, in contrast, uses probabilistic reasoning to justify the application of transformations that may change, within probabilistic accuracy bounds, the result that the program produces.
Our new approach produces probabilistic guarantees of the form $\mathbb{P}(|D| \geq B) \leq \epsilon$, $\epsilon \in (0, 1)$, where D is the difference between the results that the transformed and original programs produce, B is an acceptability bound on the absolute value of D, and ϵ is the maximum acceptable probability of observing large $|D|$. We show how to use our approach to justify the application of loop perforation (which transforms loops to execute fewer iterations) to a set of computational patterns.

1 Introduction

The standard approach to program transformation involves the use of discrete logical reasoning to prove that the applied transformation does not change the observable semantics of the program, This paper, in contrast, introduces a novel approach that uses probabilistic reasoning to justify transformations that may change the result that the program produces.

Our approach provides probabilistic guarantees that the absolute value of the difference between the results that the transformed and original programs produce will rarely be large. A user or developer can specify bounds on the acceptable difference. The analysis can then determine the conditions under which the transformed computation satisfies the probabilistic guarantees for those bounds.

1.1 Loop Perforation

In this paper, we focus on *loop perforation*, which transforms loops to execute only a subset of their original iterations. Empirical results demonstrate the utility and effectiveness of loop perforation in reducing the amount of time (and/or other resources such as energy) that the application requires to produce a result while preserving acceptable accuracy [33,18,25,38]. While investigating the reasons behind these empirical results, we identified specific computations in our benchmark applications that interacted well with loop perforation.

Inspired by these computations, in this paper we present four generalized computational patterns that interact well with loop perforation. We have previously proposed the the use of Monte-Carlo simulation to explore how loop perforation changes the result that specific computational patterns produce [35]. In this paper we propose an analytical approach that produces algebraic expressions that characterize the effect of loop perforation on our identified set of computational patterns.

1.2 Computational Patterns

We present four computational patterns that interact well with loop perforation — the *sum* pattern, which calculates the sum of elements, the *mean* pattern, which calculates the mean of elements, the *argmin-sum* pattern, which calculates the index of the minimum sum of elements, and the *ratio* pattern, which calculates the ratio of two sums. These patterns identify general classes of computations to which we can apply the analyses that we present in this paper (rather than specific syntactic structures). The following code, for example, uses a `for` loop to implement a mean pattern. Note that the pattern abstracts away the details of the computation performed in each loop iteration, represented by a function call `f(i)`. We call each such abstracted value an *input* to the pattern.

```
sum = 0.0;
for (int i = 1; i <= n; i++) sum += f(i);
mean = sum / n;
```

In general, there may be many potential ways to realize each pattern in an actual program. In some cases the pattern may be inherent in the programming language constructs used to express the computation. For example, the sum pattern may be realized by an explicit reduction operation in a functional language. In other cases (as in the `for` loop example above) the pattern may be realized by combinations of constructs. In such cases existing program analyses (for example, reduction recognition [16,20]) can identify specific instances of these patterns.

1.3 Modeling and Analysis

We quantify the effect of loop perforation by defining the *perforation noise* – the difference between the result that the perforated computation produces and the result that the original computation produces. We denote this (signed) noise as D. The probabilistic guarantees have the form: $\mathbb{P}(|D| \geq B) \leq \epsilon$, $\epsilon \in (0, 1)$, where $\mathbb{P}(|D| \geq B)$ is the probability that the absolute value $|D|$ is greater than or equal to some bound B. The probability ϵ is the maximum acceptable probability of observing large $|D|$.

We use random variables to model our *uncertainty* about the input values and the results that subcomputations produce. We express the perforation noise as a function of these random variables. Random variables in our analyses can represent 1) inherent randomness in the inputs and/or 2) our incomplete knowledge about the exact underlying processes that produce these inputs. These two forms of uncertainty are called aleatory (objective) and epistemic (subjective) uncertainty. Our probabilistic model therefore allows us to analyze both probabilistic and deterministic computations (although in this paper we focus on deterministic computations).

We use properties of random variables, in combination with the applied transformations, to derive algebraic expressions that characterize the perforation noise. Specifically, for each pattern our analysis produces algebraic expressions that characterize the expected value and variance of the perforation noise as well as the probability of observing large absolute perforation noise. Our analysis also identifies the conditions under which these expressions are accurate. These conditions may include the number of iterations of perforated loops or distribution or independence assumptions involving random variables. Multiple analyses, each with different conditions, may be applicable to a single pattern. The expressions and the conditions that the analysis produces precisely characterize the effect of loop perforation on the result that the computation produces, providing the information that automated procedures, users, or developers need to determine whether to apply a candidate transformation.

1.4 Contributions

To the best of our knowledge, this paper is the first to propose the concept of using static program analysis to derive probabilistic accuracy bounds to justify transformations that may change the result that the program produces. It is also the first to present specific static analyses which produce such probabilistic bounds. This paper makes the following specific contributions:

- **New Paradigm:** It presents a new paradigm for justifying program transformations. Instead of using discrete logical reasoning to justify transformations that preserve the exact semantics of the program, this paradigm uses probabilistic reasoning to justify transformations that may, within guaranteed probabilistic bounds, change the result that the program produces.
- **Probabilistic Modeling of Uncertainty:** Every static analysis must somehow characterize its uncertainty about the behavior of the analyzed computation. The standard approach is to use conservative, worst-case reasoning to show that the transformation preserves the semantics in all cases.
  Our novel approach, in contrast, models the values which the computation manipulates as random variables. We use properties of these random variables to reason about the effect of the transformation on the values that the computation produces. In particular, our analysis produces derived random variables that model the difference between the result that the transformed computation produces and the result that the original computation produces.
- **Probabilistic Accuracy Guarantees:** The analysis extracts properties of the derived random variables such as their mean, variance, and probabilistic bounds on their magnitude. It uses these properties to extract probabilistic accuracy guarantees that characterize the probability of observing unacceptably large changes in the result that the transformed program produces. It uses these guarantees to determine whether or not it is acceptable to apply the transformation.

– **Pattern-based Analyses:** It presents a set of computational patterns (*sum*, *mean*, *argmin-sum*, and *ratio*) with transformations that can be analyzed using probabilistic analyses. For each pattern it presents a probabilistic analysis that characterizes the effect of loop perforation on the result that the pattern produces. Each analysis produces expressions for the expected value and variance of the absolute perforation noise and bounds that characterize the probability of observing large perforation noise.

2 Example

Swaptions is a financial analysis application from the PARSEC benchmark suite [4]; it uses Monte Carlo simulation to calculate the price of a portfolio of swaption financial instruments. Figure 1(a) presents an abstract version of a perforatable computation from this application. The loop performs a sequence of simulations to compute the mean simulated value of the swaption into the variable `dMeanPrice`. The analysis recognizes this computation as an instance of the *mean* pattern.

```
float dPrice = 0.0;
for (i = 1; i <= lTrials; i += 1) {
   float simres = runSimulation(this, i)
   dPrice += simres;
}
double dMeanPrice = dPrice / lTrials;
printf("%g\n", dMeanPrice);
```

(a) Original Computation

```
float dPrice = 0.0;
for (i = 1; i <= lTrials; i += 2) {
   float simres = runSimulation(this, i)
   dPrice += simres;
}
double dMeanPrice = (dPrice * 2) / lTrials;
printf("%g\n", dMeanPrice);
```

(b) Transformed Computation

Fig. 1. Swaptions Computation: Original and Transformed Computations

Figure 1(b) presents the transformed version of the computation after loop perforation [18], applied in this case by changing the induction variable increment from `1` to `2`. The perforated loop therefore executes half of the iterations of the original loop. The transformed computation also extrapolates the result to eliminate any systematic bias introduced by executing fewer loop iterations.

Modeling Values With Random Variables. The analysis models the value of `simres` at each iteration as a random variable X_i, $i \in \{1, \ldots n\}$. The variable `dPrice` contains the sum of the X_i. In the original computation, the final value of `dPrice` is $S_O = \sum_{i=1}^{n} X_i$. In the perforated computation the final value is $S_P = \sum_{i=1}^{n/2} X_{2i-1}$ (for simplicity, we present the analysis for the case when `n` is even). After extrapolating S_P, the perforation noise $D = \frac{1}{n}\big(2S_P - S_O\big)$.

Scenarios. We have developed analyses for a variety of scenarios, with each scenario characterized by properties such as the distributions of the X_i and the number of loop iterations `n`. For each scenario the analysis produces an ϵ and B such that $\mathbb{P}(|D| \geq B) \leq \epsilon$, $\epsilon \in (0, 1)$. Specific scenarios for which analyses exist include (see Section 4) (a) the X_i have finite mean and covariance, (b) the X_i are independent and identically distributed (i.i.d.), (b.1) in addition, the number of iterations `n` of the original loop is large, or (b.2) `n` may be small but the X_i are normally distributed, or (c) the X_i are correlated and form a random walk.

Note that some scenarios are more specific than others; in general, analyses for more specific scenarios tend to deliver tighter probabilistic bounds.

Scenario (b.1). We next discuss how the analysis proceeds for Scenario (b.1) (the X_i are i.i.d. and `n` is large). Using results from Section 4, the expected value of the perforation noise $\mathbb{E}(D) = 0$, and the variance is $\mathrm{var}(D) = \sigma^2\left(\frac{1}{m} - \frac{1}{n}\right) = \frac{\sigma^2}{n}$, where σ^2 is the variance of the input variables X_i.

Because S_P and $S_O - S_P$ are sums of a large number of i.i.d. random variables, the distribution of their means approaches a normal distribution (by a Central Limit Theorem argument). Furthermore, since D is a linear combination of these two independent means, D is also normally distributed and we can use the Gaussian confidence interval to obtain the following bound: with probability at least $1 - \epsilon$, $|D| < z_{1-\frac{\epsilon}{2}}\sqrt{\mathrm{var}(D)} = z_{1-\frac{\epsilon}{2}}\frac{\sigma}{\sqrt{n}}$, where z_α is the quantile function of the standard normal distribution. For example, $\mathbb{P}(|D| \geq B) \leq 0.05$ for $B \approx \frac{1.96\sigma}{\sqrt{n}}$ ($z_{0.975} \approx 1.96$).

Comparison with Worst-Case Analysis. We next compare the probabilistic and worst-case analyses in an identical scenario. We assume that the X_i are i.i.d. random variables drawn from the uniform distribution on $[a, b]$. In this case the 95% probabilistic bound is $B_P = 1.96\frac{b-a}{\sqrt{12n}} \approx 0.6\frac{b-a}{\sqrt{n}}$ ($\sigma^2 = \frac{(b-a)^2}{12}$ for the uniform distribution). The worst-case analysis, on the other hand, extracts the bound $B_W = \frac{b-a}{2}$. Note that the worst case bound B_W is asymptotically $\sqrt{n}$ times larger than the probabilistic bound B_P.

Choosing an Appropriate Scenario. In general, the validity of the probabilistic bounds may depend on how closely the actual use case matches the selected analysis scenario. In some cases it may be appropriate to choose a scenario by recording values from instrumented representative executions of the computation, then using the values to (potentially conservatively) select aspects of the scenario such as specific probability distributions or the expected number of loop iterations [24]. In other cases it may be appropriate to have a user or developer simply provide this information directly to the analysis [24].

Identifying Appropriate ϵ and B. In general, acceptable values for ϵ and B will depend on the application and the context in which it is used. We therefore anticipate that the user (or potentially the developer) will identify the maximum acceptable ϵ and B. Transformations with ϵ and B less than these maxima are acceptable. Transformations with ϵ or B greater than these maxima are unacceptable.

3 Preliminaries

We next describe the notation that we use throughout the probabilistic analyses. The original loop executes n iterations; the perforated loop executes m, $m < n$ iterations. The *perforation rate* $r = 1 - \lfloor \frac{m}{n} \rfloor$. A loop perforation *strategy* can be represented by a $n \times 1$ *perforation vector* P, each element of which corresponds to a single loop iteration. The number of non-zero elements of P is equal to

m. The all-ones perforation vector $A = (1, \dots, 1)'$ represents the original (non-perforated) computation. Some transformations may use the perforation vector. The perforation transformation for the sum pattern, for example, uses the values of the non-zero elements of the vector P to extrapolate the final result.

Interleaving perforation executes every k-th iteration, where $k = \lfloor \frac{n}{m} \rfloor$. The corresponding perforation vector has elements $P_{ki+1} = 1$, where $i \in \{0, ...m-1\}$, and $P_{ki+j} = 0$ for $j < k, j \neq 1$. *Truncation perforation* executes m iterations at the beginning of the loop; the perforation vector has elements $P_i = 1$ for $1 \leq i \leq m$, and $P_i = 0$ otherwise. *Randomized perforation* selects a random subset of m elements. All of these perforation strategies can be implemented efficiently without explicitly creating the vector P.

4 Patterns and Analyses

For each pattern we present an example of the original and the transformed code. For simplicity we apply an interleaving perforation and assume a number of iterations `n` to be a multiple of the new loop increment `k`.

In each pattern analysis section we first present the assumptions we make on the distribution of the inputs. These assumptions characterize our uncertainty about the values of these inputs. With these assumptions in place, we derive expressions for 1) the mean perforation noise, 2) the variance of the perforation noise, and 3) bounds on the probability of observing large absolute perforation noise. In some cases we perform additional analyses based on additional assumptions. When applicable, we present bounds based on Chebyshev's inequality, Hoeffding's inequality, and Gaussian confidence intervals. We present a more detailed derivation of these expressions in our accompanying technical report [24].

4.1 Sum Pattern

We present an example of the original and perforated code for the extrapolated sum pattern in Figure 2. We first present a generalized analysis for the sum of correlated random variables. We then present specializations of the analysis under additional assumptions. Special cases that we analyze include independent and identically distributed (i.i.d.) inputs and inputs generated by a random walk.

Original code

```
double sum = 0.0;
for (int i = 1; i <= n; i++) {
  sum += f(i);
}
```

Transformed Code

```
double sum = 0.0;
for (int i = 1; i <= n; i+=k) {
  sum += f(i);
}
sum *= k;
```

Fig. 2. Sum Pattern; Original and Transformed Code

Assumptions. We first assume only that the terms of the sum have a common finite mean μ and finite covariance.

Analysis. For $i = 1, \dots, n$, let $X_i = \texttt{f}(i)$ be the i-th term of the summation. We model our uncertainty about the values X_i by treating $X = (X_1, \dots, X_n)'$ as a vector of n random variables with mean μ and covariance matrix Σ with elements $(\Sigma)_{ij} = \mathrm{cov}(X_i, X_j)$. Let A be the all-ones vector defined in Section 3, then $A'X = \sum_{i=1}^{n} X_i$. Let P be a perforation vector with m non-zero elements. Then $P'X = \sum_{i=1}^{n} P_i X_i$ is the result of the perforated computation. The signed perforation noise is $D \equiv P'X - A'X = (P - A)'\,X$ with

$$\mathbb{E}(D) = \mu \sum_{i=1}^{n} (P_i - 1), \tag{1}$$

$$\mathrm{var}(D) = \sum_{i,j} (P_i - 1)\,(P_j - 1)\,\Sigma_{i,j}. \tag{2}$$

To avoid systematic bias, we can choose P so that $\mathbb{E}(D) = 0$. In particular, it follows from Equation 1 that an estimate is *unbiased* if and only if $\sum_{i=1}^{n} P_i = n$. One extrapolation strategy equally extrapolates every non-zero element, choosing $P_i = \frac{n}{m}$ for non-zero elements P_i.

If P satisfies $\mathbb{E}(D) = 0$, we can use Chebyshev's inequality and $\mathrm{var}(D)$ to bound the absolute perforation noise, such that, with probability at least $1 - \epsilon$

$$|D| < \sqrt{\frac{\mathrm{var}(D)}{\epsilon}} \tag{3}$$

This bound will be conservative in practice; additional knowledge (e.g., independence or distribution of X_i) can be used to derive tighter bounds. We next study a number of special cases in which additional assumptions enable us to better characterize the effect of perforation.

Independent Variables

Assumptions. We assume that the elements $X_i = \texttt{f}(i)$ of the summation are i.i.d. random variables with finite mean μ and variance σ^2. To derive a tighter bound on the mean and the variance of the absolute perforation noise, we consider additional assumptions – specifically, that the X_i are normally distributed, or that the X_i are bounded.

Analysis. From (1), we know that $\mathbb{E}(D) = 0$ for any perforation P such that $\sum_i P = n$. From (2), and since the covariance matrix Σ of i.i.d. variables has non-zero values only along its leading diagonal, it follows that $\mathrm{var}(D) = \sigma^2 \sum_i (1 - P_i)^2$. It is straightforward to show that this value is minimized by any perforation vector P with $n - m$ zeros and the remaining elements taking the value $\frac{n}{m}$. In this case, the variance takes the value

$$\mathrm{var}(D) = \frac{\sigma^2\, n\,(n - m)}{m}. \tag{4}$$

We can immediately bound the probability of observing large absolute perforation noise using Chebyshev's inequality (Equation 3).

We can get potentially tighter bounds if we make additional assumptions. If we assume that each term X_i is normally distributed, then D will also be normally distributed. Consequently, $\mathbb{E}(D) = 0$ and $\mathrm{var}(D)$ remains the same as in Equation 4.

The normality of D allows us to obtain a tighter bound on the perforation noise. In particular, with probability $1 - \epsilon$

$$|D| \leq z_{1-\frac{\epsilon}{2}} \sqrt{\mathrm{var}(D)} \tag{5}$$

where z_α is the quantile function of the standard normal distribution. As a comparison, for $\epsilon = 0.01$ the bound (5) is 6.6 times smaller than the Chebyshev-style bound (3). For normally distributed D we can also bound the absolute perforation noise. In particular, $|D|$ has a half-normal distribution with mean $\mathbb{E}\Big(|D|\Big) = \sigma\sqrt{\frac{2n(n-m)}{\pi m}}$, and variance $\mathrm{var}(|D|) = \Big(1 - \frac{2}{\pi}\Big)\,\mathrm{var}(D)$.

If, instead, we assume that each X_i is bounded, falling in the range $[a, b]$, we can apply Hoeffding's inequality to bound the absolute perforation noise $|D|$. Let $X'_i = (P_i - 1)X_i$, and note that the variables X'_i are also mutually independent. The range of X'_i is $[a_i, b_i] = \big[(P_i - 1)a, (P_i - 1)b\big]$. Then the sum $\sum_{i=1}^{n}(b_i - a_i)^2 = (b - a)^2 \frac{n\,(n-m)}{m}$, and thus, with probability at least $1 - \epsilon$

$$|D| < \sqrt{\frac{1}{2} \ln \frac{2}{\epsilon} \cdot \sum_{i=1}^{n} \Big(b_i - a_i\Big)^2} = (b - a)\sqrt{\frac{n\,(n-m)}{2m} \ln \frac{2}{\epsilon}}. \tag{6}$$

Nested Loops. We next extend the sum analysis (for i.i.d. inputs) to nested loops. The outer loop executes n_1 iterations (m_1 iterations after perforation); the inner loop executes n_2 iterations (m_2 iterations after perforation). When both loops are perforated, $\mathbb{E}(D) = 0$. We use Equation 4 to compute $\mathrm{var}(D)$ by assigning $n = n_1 n_2$ and $m = m_1 m_2$. The result generalizes to more deeply nested loops.

Random Walk

Assumptions. We assume that the sequence X of random variables is a random walk with independent increments. Specifically, we assume that the sequence is a Markov process, and that the differences between the values at adjacent time steps $\delta_i = X_{i+1} - X_i$ are a sequence of i.i.d. random variables with mean 0 and variance σ^2. Let $X_0 = \mu$ be a constant.

Analysis. From the assumption $\mathbb{E}(\delta_i) = 0$, it follows by induction that the expected value of every element is $\mathbb{E}(X_i) = \mu$. As a consequence, for any perforation vector that satisfies $\sum_{i=1}^{n} P_i = n$, we have that $\mathbb{E}(D) = 0$.

For $i < j$, the covariance between X_i and X_j satisfies $cov(X_i, X_j) = i\sigma^2$. Therefore, the covariance matrix Σ has entries $(\Sigma)_{ij} = \sigma^2 \min\{i, j\}$, and the variance of the perforation noise satisfies

$$\mathrm{var}(D) = \sigma^2 \sum_{i,j} (1 - P_i)\,(1 - P_j) \min\{i, j\}. \tag{7}$$

We may choose a perforation strategy P by minimizing this variance (and thus minimizing Chebyshev's bound on the absolute perforation noise). For example, when $P_i = 2$ for odd i and 0 otherwise, we have that $\mathrm{var}(D) = \frac{n}{2}\sigma^2$. Once again, we can use Chebyshev's inequality or Gaussian confidence intervals to derive a probabilistic accuracy bound.

4.2 Mean Pattern

We present an example of the original and perforated code for the mean pattern in Figure 3.

Original code	Transformed Code
`double sum = 0.0;`	`double sum = 0.0;`
`for (int i = 1; i <= n; i++) {`	`for (int i = 1; i <= n; i+=k) {`
`  sum += f(i);`	`  sum += f(i);`
`}`	`}`
`double mean = sum / n;`	`double mean = sum * k / n;`

Fig. 3. Mean Pattern; Original and Transformed Code

We can extend the analysis for the sum pattern (Section 4.1) because the result of the mean computation is equal to the result of the sum computation divided by n. We denote the perforation noise of the sum as D_{Sum}, the output produced by the original computation as $\frac{1}{n}A'X$, and the output produced by the perforated computation as $\frac{1}{n}P'X$. The perforation noise of the mean D in the general case with correlated variables is $D \equiv \frac{1}{n}\big(P'X - A'X\big) = \frac{1}{n}D_{Sum}$. By the linearity of expectation, the perforation noise has expectation $\mathbb{E}(D) = \frac{1}{n}\mathbb{E}(D_{Sum})$ and variance

$$\mathrm{var}(D) = \frac{1}{n^2}\,\mathrm{var}(D_{Sum}). \tag{8}$$

The derivation of the bounds for the more specific cases (i.i.d., normal, random walk inputs) is analogous to the derivation discussed in Section 4.1. In particular if we assume i.i.d. inputs, the variance $\mathrm{var}(D) = \sigma^2\big(\frac{1}{m} - \frac{1}{n}\big)$. Based on Chebyshev's and Hoeffding's inequalities, we can derive algebraic expressions that characterize the probabilistic accuracy bounds for this case. A similar result can be shown for the random walk case.

We can also obtain a potentially tighter Gaussian interval style bound if we assume a large number of i.i.d. inputs with finite mean and variance. In this case the sums $P'X$ and $(A - \frac{m}{n}P)'X$ will be independent and their means will be approximately normally distributed (by a Central Limit Theorem argument).[1] Consequently, the perforation noise D, which is a linear combination of these two means, will also be approximately normally distributed. We can then use Equation 5 to calculate a bound on the perforation noise.

[1] Note that the tails of the distribution of the mean (which we use to bound the perforation noise) converge to the normal distribution slower than the means. The Berry-Esseen inequality can be used to determine how closely the normal distribution approximates the actual distribution of the sum. In particular, if n is the number of the terms, the maximum distance between the standardized sum distribution and the normal distribution is less than $\delta \approx 0.48\frac{\rho}{\sigma^3\sqrt{n}}$, where $\rho = \mathbb{E}(|X_i|^3)$ and $\sigma^2 = \mathrm{var}(X_i)$.

4.3 Argmin-Sum Pattern

We present an example of the original and transformed code for the argmin-sum pattern in Figure 4.[2]

Original code	Transformed Code
`double best = MAX_DOUBLE;`	`double best = MAX_DOUBLE;`
`int best_index = -1;`	`int best_index = -1;`
`for (int i = 1; i <= L; i++) {`	`for (int i = 1; i <= L; i++) {`
`  s[i] = 0;`	`  s[i] = 0;`
`  for (int j = 1; j <= n; j++)`	`  for (int j = 1; j <= n; j+=k)`
`    s[i] += f(i,j);`	`    s[i] += f(i,j);`
`  if (s[i] < best) {`	`  if (s[i] < best) {`
`     best = s[i];`	`     best = s[i];`
`     best_index = i;`	`     best_index = i;`
`  }`	`  }`
`}`	`}`
`return best_index;`	`return best_index;`

Fig. 4. Argmin-Sum Pattern; Original and Transformed Code

Assumptions. For each $i \in \{1, \ldots, L\}$, we assume that $X_{i,j} = \mathtt{f(i,j)}$ are independent and drawn from some distribution F. The elements of the perforation vector P take only the values from the set $\{0, 1\}$.

Analysis The output of the argmin-sum pattern is an index which is used later in the program. To calculate the perforation noise, we model the *weight* of an index i as the entire sum $X_i = \sum_{j=1}^{n} X_{i,j}$. Therefore, the original computation produces the value $S_O = \min_i A'X_i = \min_i \sum_{j=1}^{n} X_{i,j}$, while the perforated computation produces the value $S_P = \sum_{j=1}^{n} X_{\gamma,j}$, where $\gamma \equiv \arg\min_i \sum_{j=1}^{m} X_{i,j}$ and m is the reduced number of steps in the perforated sum. Note that the independence of the variables $X_{i,j}$ implies that we can, without loss of generality, choose any perforation strategy with perforation rate r.

We are interested in studying the perforation noise $D \equiv S_P - S_O$. Note that the perforation noise D is non-negative because S_O is a minimum sum, and so $D = |D|$ is also the absolute perforation noise.

Let $Y_i \equiv \sum_{j=1}^{m} X_{i,j}$ and $Z_i \equiv \sum_{j=m+1}^{n} X_{i,j}$. Then, $S_O = \min_i (Y_i + Z_i) = Y_\omega + Z_\omega$ and $S_P = Y_\gamma + Z_\gamma$ where $\gamma = \arg\min_i Y_i$ and $\omega = \arg\min_i (Y_i + Z_i)$ is the index of the minimum sum. Then the perforation noise satisfies

$$D \leq Z_\gamma - \min_i Z_i. \tag{9}$$

Let $\bar{D} \equiv Z_\gamma - \min_i Z_i$ denote this upper bound. We can obtain conservative estimates of the perforation noise D by studying $\bar{D}$. Note that for this pattern, $\bar{D}$ is always non-negative (because $Z_\gamma \geq \min_i Z_i$).

[2] We can apply the same analysis to the *min-sum* pattern, which returns the (extrapolated) value `best` instead of the index `best_index`. It is also possible to modify this analysis to support the *max-sum* and *argmax-sum* patterns.

Let Z be a sum of $n-m$ independent F-distributed random variables. Then (1) Z_i has the same distribution as Z, (2) γ is independent of Z_γ, and (3) Z_γ has the same distribution as Z. Therefore,

$$\mathbb{E}(\bar{D}) = \mathbb{E}(Z) - \mathbb{E}(\min_i Z_i), \tag{10}$$

or, put simply, the expectation of our bound $\bar{D}$ is the difference between the mean of Z and its first order statistic (given a size L sample).

To proceed with the analysis, we make the additional assumption that Z is uniformly distributed on the interval $a \pm \frac{w}{2}$ of width $w > 0$ and center a.[3] Let Z_i be i.i.d. copies of Z.

Define $M_L = \min_{i \le L} Z_i$. Then $\frac{1}{w}(M_L - a + \frac{w}{2})$ has a Beta$(1, L)$ distribution, and so $\mathbb{E}(M_L) = a + \frac{w}{L+1} - \frac{w}{2}$ and variance $\mathrm{var}(M_L) = \frac{Lw^2}{(L+1)^2(L+2)}$. From (10), we have $\mathbb{E}(\bar{D}) = \frac{w}{2} - \frac{w}{L+1}$. Furthermore, as γ is independent of every Z_i, it follows that Z_γ is independent of M_L. Therefore,

$$\mathrm{var}(\bar{D}) = \frac{1}{12}w^2 + \frac{Lw^2}{(L+1)^2(L+2)}. \tag{11}$$

The mean and variance of $\bar{D}$ can be used to derive one-sided Chebyshev style bounds on $\bar{D}$ and, since $|D| = D < \bar{D}$, bounds on the absolute perforation noise $|D|$. In particular, using Chebyshev's one-sided inequality, it follows that with probability at least $1 - \epsilon$

$$|D| < \sqrt{\mathrm{var}(\bar{D})\Big(\frac{1}{\epsilon} - 1\Big)} + \mathbb{E}\bar{D} \tag{12}$$

4.4 Ratio Pattern

We present an example of the original and transformed code for the ratio pattern in Figure 5.

Original code	Transformed Code
`double numer = 0.0;`	`double numer = 0.0;`
`double denom = 0.0;`	`double denom = 0.0;`
`for (int i = 1; i <= n; i++) {`	`for (int i = 1; i <= n; i+=k) {`
`  numer += x(i);`	`  numer += x(i);`
`  denom += y(i);`	`  denom += y(i);`
`}`	`}`
`return numer/denom;`	`return numer/denom;`

Fig. 5. Ratio Pattern; Original and Transformed Code

[3] We anticipate that Z will in practice rarely be uniform, however this assumption simplifies the analysis and is in some sense conservative if we choose the center and width to cover all but a tiny fraction of the mass of the true distribution of Z. Note that when, instead, Z is Gaussian, the variance of the perforation noise does not have a closed form [2]. However, if we assume Z to be uniform, we might take our approximation to cover some number of standard deviations.

Assumptions. Let $X_i = \mathtt{x}(i)$ and $Y_i = \mathtt{y}(i)$ denote random variables representing the values of the inner computations. We assume that the sequence of pairs (X_i, Y_i) are i.i.d. copies of a pair of random variables (X, Y), where $Y > 0$ almost surely. Define $Z = X/Y$ and $Z_i = X_i/Y_i$. For some constants μ and σ_Z^2, we assume that the conditional expectation of Z given Y is μ, i.e., $\mathbb{E}(Z|Y) = \mu$, and that the conditional variance satisfies $\mathrm{var}(Z|Y) = \frac{\sigma_Z^2}{Y}$.

Analysis. The elements of the perforation vector P only take values from the set $\{0, 1\}$. Note that the independence of the pairs (X_i, Y_i) from different iterations implies that the perforation strategy does not influence the final result. To simplify the derivation, but without loss of generality, we use the perforation vector P in which the first m elements are 1 and the remaining elements 0.

Define $Y_1^n = A'Y = \sum_{i=1}^n Y_i$ and $Y_1^m = P'Y = \sum_{i=1}^m Y_i$ and define X_1^n and X_1^m analogously. Then the value of the original computation is $S_O = \frac{X_1^n}{Y_1^n} = \sum_{i=1}^n \frac{Y_i}{Y_1^n} Z_i$, while the value of the perforated computation is given by $S_P = \sum_{i=1}^m \frac{Y_i}{Y_1^m} Z_i$, where m is the reduced number of steps in the perforated sum. Note that in the previous equations, we used the identity $X_i = Y_i Z_i$.

We begin by studying the (signed) perforation noise $D \equiv S_P - S_O$. The conditional expectation of D given $Y_{1:n} = \{Y_1, \dots, Y_n\}$ satisfies $\mathbb{E}(D|Y_{1:n}) = \sum_{i=1}^n \frac{Y_i}{Y_1^n}\mu - \sum_{i=1}^m \frac{Y_i}{Y_1^m}\mu = 0$. The conditional variance satisfies $\mathrm{var}(D|Y_{1:n}) = \sigma_Z^2 \left(\frac{1}{Y_1^m} - \frac{1}{Y_1^n} \right)$ By the law of iterated expectations $\mathbb{E}(D) = \mathbb{E}(\mathbb{E}(D|Y_{1:n})) = 0$.

To proceed with an analysis of the variance of the perforation noise D, we make a distributional assumption on Y. In particular, we assume that Y is gamma distributed with shape $\alpha > 1$ and scale $\theta > 0$. Therefore, the sum Y_1^m also has a gamma distribution with parameters $\alpha' = m\alpha$, $\theta' = \theta$, $\frac{1}{Y_1^m}$ has an inverse gamma distribution with mean $(\theta(m\alpha - 1))^{-1}$, and so

$$\mathrm{var}(D) = \frac{\sigma_Z^2}{\theta} \left(\frac{1}{m\alpha - 1} - \frac{1}{n\alpha - 1} \right). \tag{13}$$

Again, using Chebyshev's inequality, we can bound the probability of large absolute perforation noise $|D|$.

5 Discussion

Usage Scenarios. We anticipate several usage scenarios for the analyses we present in Section 4. First, the analyses can provide the formal foundation required to justify the automatic application of loop perforation. In this scenario, the analysis works with a probabilistic accuracy specification (which provides the desired accuracy bounds and probability with which the transformed computation should satisfy the bounds) and a specification of the probability distributions for the random variables used to model pattern inputs. These probability distributions can be provided either by a developer, by a user, or by fitting distributions to values observed during profiling executions. In [24] we present an initial empirical evaluation of our probabilistic analyses on perforatable computations from the PARSEC benchmark suite.

Second, the analyses can also help users and developers better understand the effect of loop perforation. They may then use this information to select an optimal operating point for their application given their combined performance and accuracy constraints and requirements.

Third, the analyses can also help a control system dynamically select optimal application operating points as underlying characteristics (such as load, clock rate, number of processors executing the computation, or any other characteristic that many affect the delivered computational capacity) of the underlying computational platform change [18,33].

In all of these scenarios the probabilistic analyses in this paper can be used to better understand the shape of the trade-off space and more productively drive the selection of perforation policies, with appropriate maximum acceptable ϵ and B determining the range of available policies.

Scope. In this paper we provide probabilistic guarantees for the accuracy of perforated computations. We expect that the basic framework of the probabilistic guarantees (algebraic expressions for expected values, variances, and probabilistic bounds) will remain largely the same for other transformations (the derivation of the expressions will of course differ). We note that even for loop perforation, we do not attempt to provide an exhaustive list of the possible patterns and analyses. The statistical literature provides a comprehensive treatment of operations on random variables [41] and order statistics of random variables [2]. The basic compositional properties of probability distributions under such operations can provide the foundation for the analysis of computations which employ many of these operations. In addition, for the random perforation strategy, survey sampling [9] can provide useful bounds that do not make assumptions on the distribution of the inputs for a number of aggregation computations.

We note that, given known composition properties of operations on probability distributions (for example, sums of Gaussian distributions are Gaussian; multiplying a Gaussian by constant produces another Gaussian), it is possible to compose our pattern-based analyses in straightforward ways to analyze more complex computations. For example, it is straightforward to generalize the analysis of the sum pattern to analyze arbitrary linear computations over values modeled using Gaussian distributions.

We also anticipate that a number of program analyses or type systems can work in concert with our probabilistic analyses. These analyses and type systems can, for example, help identify computational patterns, increase confidence in some of the input assumptions, or reason about safety of the transformation. For example, analyses or type systems may distinguish critical parts of the computation (which, if transformed, can dramatically change the behavior of the application and as such should not be perforated), from approximate parts of the computation, which can be perforated [5,37].

6 Related Work

Loop Perforation and Task Skipping: Loop perforation [18,25,38] can be seen as a special case of task skipping [33,34]. The first publication on task skipping used linear regression to obtain empirical statistical models of the time and accuracy effects of skipping tasks and identified the use of these models in purposefully skipping tasks to reduce the amount of resources required to perform the computation while preserving acceptable accuracy [33].

The first publication on loop perforation presented a purely empirical justification of loop perforation with no formal statistical, probabilistic, or discrete logical reasoning used to justify the transformation [18]. The first statistical justification of loop perforation used Monte Carlo simulation [35]. The first probabilistic justification for loop perforation used a pattern-based static analysis and also presented the use of profiling runs on representative inputs and developer specifications to obtain the required probability distribution information [24]. The probabilistic analyses in this paper can help users or developers better understand the shape of the induced accuracy vs. performance trade-off space and select optimal operating points within this space given their combined accuracy and performance requirements and/or constraints.

Continuity, Sensitivity, and Robustness: Chaudhuri et al. present a program analysis for automatically determining whether a function is continuous [6]. The reasoning is deterministic and worst-case. An extension of this research introduces a notion of function robustness, and, under an input locality condition, presents an approximate memoization approach similar to loop perforation [7]. For a special case when the inputs form a Gaussian random walk (as described in Section 4.1) and the loop body is a robust function, the paper derives a probabilistic bound to provide a justification for applying loop perforation.

Reed and Pierce present a type system for capturing function sensitivity, which measures how much a function may magnify changes to its inputs [32]. Although the language contains probabilistic constructs, the type system uses deterministic worst-case reasoning, resulting in a worst-case sensitivity bound.

Modeling Uncertainty. Typical approaches for modeling uncertainty involve the use of intervals, random variables, or fuzzy sets to represent values, and the definition of computational patterns that operate on these uncertain values. Interval analysis [28] represents uncertain values as intervals and defines basic arithmetic operations on such values. It is often used to analyze the worst-case rounding error in numerical computations, ideally producing small interval sizes. For loop perforation the interval sizes are typically much larger and the derived bounds therefore much less precise.

Additional knowledge about the inputs can make it possible to use probabilistic, fuzzy, or hybrid modeling of the computations [21] to provide tighter bounds. In this paper we use random variables to model uncertainty. The source of this uncertainty can be either 1) innate randomness in the inputs or computation or 2) our partial understanding of parts of the computation. Fuzzy or hybrid approaches to modeling uncertainty may also, at least in principle, provide alternate justifications for loop perforation.

Probabilistic Languages and Analyses: Researchers have previously defined languages for probabilistic modeling, in which programs work directly with probability distributions [36,22,19,31,11,30,13], and analyses to reason about probabilistic programs [29,12,26,27,23,10,39]. Researchers have also used a probabilistic foundation to quantitatively reason about certain properties of deterministic programs [14,8]. Our approach quantitatively analyzes the application of loop perforation to a set of amenable computational patterns, which may appear in deterministic or probabilistic programs. It specifies probabilistic semantics at a pattern level instead of the statement level. In comparison with general probabilistic analyses, pattern-based analyses can, typically, provide more precise accuracy bounds, since patterns provide additional information about the nature of the analyzed computations (instances of patterns). Moreover, patterns can identify additional information such as a definition of perforation noise (e.g., for the argmin-sum pattern), which may be impossible for a general probabilistic semantics to capture. Computational patterns similar to ours have also been used to provide more precise analyses in other contexts [15].

Performance vs. Accuracy Trade-Off Management: Both task skipping [33,34] and loop perforation [18,38] can augment an application with the ability to operate at multiple points in an underlying accuracy vs. performance trade-off space. Of particular interest is the ability to move between points in this space as the application is executing, enabling the application to adapt to the underlying computing environment [18,33,34]. The empirical discovery of Pareto-optimal combinations of perforated computations can enable a control system to find and exploit optimal operating points within the trade-off space [18].

Dynamic Knobs converts static application configuration parameters into dynamic control variables, which the system can use to change the point in the underlying trade-off space at which the application executes [17]. Eon [40], Green [3], and Petabricks [1] allow developers to provide multiple implementations of a specific piece of application functionality, with each implementation potentially exhibiting different performance versus accuracy trade-offs. There is no explicit reasoning to justify the acceptability of the different alternatives – all of these systems empirically evaluate the alternatives and ultimately rely on the developer to identify only acceptable implementations.

7 Conclusion

Traditional program analysis and transformation approaches use worst-case logical reasoning to justify the application of transformations that do not change the result that the program produces. We propose instead to use probabilistic reasoning to justify the application of transformations that may, within probabilistic bounds, change the result that the program produces. A goal is to provide a reasoning foundation that can enable the application of a richer class of program transformations.

Our results demonstrate how to apply this approach to justify the use of loop perforation, which transforms the program to skip loop iterations. We identify computations that interact well with loop perforation and show how to use probabilistic reasoning to bound how much loop perforation may change the result that the program produces. This reasoning can provide the foundation required to understand, predict, and therefore justify the application of loop perforation. In the future, we anticipate the use of similar probabilistic reasoning to justify the application of a broad range of new transformations that may change the result that the program produces.

Acknowledgements. We would like to thank Michael Carbin and the anonymous reviewers for their useful comments on the earlier drafts of this paper.

This research was supported in part by the National Science Foundation (Grants CCF-0811397, CCF-0905244, CCF-1036241 and IIS-0835652) and the United States Department of Energy (Grant DE-SC0005288).

References

1. J. Ansel, C. Chan, Y. Wong, M. Olszewski, Q. Zhao, A. Edelman, and S. Amarasinghe. Petabricks: A language and compiler for algorithmic choice. In *PLDI '10*.
2. B. Arnold, N. Balakrishnan, and H. Nagaraja. *A first course in order statistics*. Society for Industrial Mathematics, 2008.
3. W. Baek and T. Chilimbi. Green: A framework for supporting energy-conscious programming using controlled approximation. In *PLDI '10*.
4. C. Bienia, S. Kumar, J. P. Singh, and K. Li. The PARSEC benchmark suite: Characterization and architectural implications. In *PACT '08*.
5. M. Carbin and M. Rinard. Automatically Identifying Critical Input Regions and Code in Applications. In *ISSTA '10*.
6. S. Chaudhuri, S. Gulwani, and R. Lublinerman. Continuity analysis of programs. In *POPL '10*.
7. S. Chaudhuri, S. Gulwani, R. Lublinerman, and S. Navidpour. Proving Programs Robust. In *FSE '11*.
8. S. Chaudhuri and A. Solar-Lezama. Smooth interpretation. In *PLDI '10*.
9. W. G. Cochran. *Sampling techniques*. John Wiley & Sons, 1977.
10. A. Di Pierro, C. Hankin, and H. Wiklicky. A systematic approach to probabilistic pointer analysis. In *ASPLAS '07*.
11. A. Di Pierro, C. Hankin, and H. Wiklicky. Probabilistic λ-calculus and quantitative program analysis. *Journal of Logic and Computation*, 2005.
12. A. Di Pierro and H. Wiklicky. Concurrent constraint programming: Towards probabilistic abstract interpretation. In *PPDP '00*.
13. N. Goodman, V. Mansinghka, D. Roy, K. Bonawitz, and J. Tenenbaum. Church: a language for generative models. In *UAI '08*.
14. S. Gulwani and G. C. Necula. Precise interprocedural analysis using random interpretation. In *POPL '05*.
15. S. Gulwani and F. Zuleger. The reachability-bound problem. In *PLDI '10*.
16. M. Hall, B. Murphy, S. Amarasinghe, S. Liao, and M. Lam. Interprocedural analysis for parallelization. *Languages and Compilers for Parallel Computing*, 1996.
17. H. Hoffman, S. Sidiroglou, M. Carbin, S. Misailovic, A. Agarwal, and M. Rinard. Dynamic knobs for power-aware computing. In *ASPLOS '11*.
18. H. Hoffmann, S. Misailovic, S. Sidiroglou, A. Agarwal, and M. Rinard. Using Code Perforation to Improve Performance, Reduce Energy Consumption, and Respond to Failures . Technical Report MIT-CSAIL-TR-2009-042, 2009.
19. J. Hurd. A formal approach to probabilistic termination. In *TPHOLs '02*.

20. K. Kennedy and J. R. Allen. *Optimizing compilers for modern architectures: a dependence-based approach.* Morgan Kaufman, 2002.
21. G. Klir. *Uncertainty and information.* John Wiley & Sons, 2006.
22. D. Kozen. Semantics of probabilistic programs. *Journal of Computer and System Sciences*, 1981.
23. M. Kwiatkowska, G. Norman, and D. Parker. Prism: Probabilistic symbolic model checker. *Computer Performance Evaluation: Modelling Techniques and Tools*, 2002.
24. S. Misailovic, D. Roy, and M. Rinard. Probabilistic and Statistical Analysis of Perforated Patterns. Technical Report MIT-CSAIL-TR-2011-003, MIT, 2011.
25. S. Misailovic, S. Sidiroglou, H. Hoffmann, and M. Rinard. Quality of service profiling. In *ICSE '10*.
26. D. Monniaux. Abstract interpretation of probabilistic semantics. In *SAS '00*.
27. D. Monniaux. An abstract monte-carlo method for the analysis of probabilistic programs. In *POPL '01*.
28. R. Moore. *Interval analysis.* Prentice-Hall, 1966.
29. C. Morgan and A. McIver. pGCL: formal reasoning for random algorithms. *South African Computer Journal*, 22, 1999.
30. S. Park, F. Pfenning, and S. Thrun. A probabilistic language based upon sampling functions. In *POPL '05*.
31. N. Ramsey and A. Pfeffer. Stochastic lambda calculus and monads of probability distributions. In *POPL '02*.
32. J. Reed and B. C. Pierce. Distance makes the types grow stronger: a calculus for differential privacy. In *ICFP '10*.
33. M. Rinard. Probabilistic accuracy bounds for fault-tolerant computations that discard tasks. In *ICS '06*.
34. M. Rinard. Using early phase termination to eliminate load imbalances at barrier synchronization points. In *OOPSLA '07*.
35. M. Rinard, H. Hoffmann, S. Misailovic, and S. Sidiroglou. Patterns and statistical analysis for understanding reduced resource computing. In *Onward! '10*.
36. N. Saheb-Djahromi. Probabilistic LCF. In *MFCS '78*.
37. A. Sampson, W. Dietl, E. Fortuna, D. Gnanapragasam, L. Ceze, and D. Grossman. Enerj: Approximate data types for safe and general low-power computation. In *PLDI '11*.
38. S. Sidiroglou, S. Misailovic, H. Hoffmann, and M. Rinard. Managing Performance vs. Accuracy Trade-offs With Loop Perforation. In *FSE '11*.
39. M. Smith. Probabilistic abstract interpretation of imperative programs using truncated normal distributions. *Electronic Notes in Theoretical Computer Science*, 2008.
40. J. Sorber, A. Kostadinov, M. Garber, M. Brennan, M. D. Corner, and E. D. Berger. Eon: a language and runtime system for perpetual systems. In *SenSys '07*.
41. M. Springer. *The algebra of random variables.* John Wiley & Sons, 1979.