Computer Science and Artificial Intelligence Laboratory

Technical Report

MIT-CSAIL-TR-2011-003 January 19, 2011

Probabilistic and Statistical Analysis of Perforated Patterns

Sasa Misailovic, Daniel M. Roy, and Martin Rinard

massachusetts institute of technology, cambridge, ma 02139 usa — www.csail.mit.edu

Probabilistic and Statistical Analysis of Perforated Patterns

Sasa Misailovic
MIT CSAIL/EECS
misailo@csail.mit.edu

Daniel M. Roy
MIT CSAIL/EECS
droy@csail.mit.edu

Martin Rinard
MIT CSAIL/EECS
rinard@csail.mit.edu

Abstract

We present a new foundation for the analysis and transformation of computer programs. Standard approaches involve the use of logical reasoning to prove that the applied transformation does not change the observable semantics of the program. Our approach, in contrast, uses probabilistic and statistical reasoning to justify the application of transformations that may change, within probabilistic bounds, the result that the program produces.

Loop perforation transforms loops to execute fewer iterations. We show how to use our basic approach to justify the application of loop perforation to a set of computational patterns. Empirical results from computations drawn from the PARSEC benchmark suite demonstrate that these computational patterns occur in practice. We also outline a specification methodology that enables the transformation of subcomputations and discuss how to automate the approach.

1. Introduction

The standard approach to program transformation involves the use of logical reasoning to prove that the applied transformation does not change the observable semantics of the program. This technical report, in contrast, introduces a novel approach that uses probabilistic and statistical reasoning to justify transformations that may change the result that the program produces. We call the resulting class of transformations *rich* transformations because of the broader scope they have to transformation the program (in comparison to the standard class of *impoverished* transformations, which always preserve the program semantics).

Our approach is designed to provide probabilistic guarantees of the following form: $\mathbb{P}(|D| \geq B) \leq \epsilon, \epsilon \in (0, 1)$, where $|D|$ is the absolute value of the difference between the result that the original program produces and the result that the transformed program produces and $\mathbb{P}(|D| \geq B)$ is the probability that $|D|$ is greater than or equal to some bound B. Users specify ϵ and B. The transformation system produces programs designed to satisfy the probabilistic guarantee for those ϵ and B.

Our approach is also designed to provide probabilistic guarantees of the form $\mathbb{E}(|D|) \leq B_\mu$ and $\mathrm{var}(|D|) \leq B_{\sigma^2}$, where $\mathbb{E}(|D|)$ is the expected value of $|D|$, B_μ is a user-specified bound on this expected value, $\mathrm{var}(|D|)$ is the variance of $|D|$, and B_{σ^2} is a user-specified bound on this variance.

1.1 Example

Consider the example program in Figure 1, which computes the sum of a set of numbers. Next consider the transformed program in Figure 2, which computes the sum of every other number in the set, then uses extrapolation to approximate the sum of all of the numbers in the set.

Like many sublinear algorithms [17], the transformed program operates on a subset of the numbers from the original program, then uses extrapolation to reconstruct an approximation to the original result. We call the transformation we use to modify the original program *loop perforation* because it perforates the loop by skipping some of its iterations.

```
double sum = 0.0;
for (int i = 0; i < n; i++) {
  sum += f(i);
}
```

Figure 1. Original Program

```
double sum = 0.0;
for (int i = 0; i < n; i += 2) {
  sum += f(i);
}
sum = sum * 2;
```

Figure 2. Transformed Program

1.1.1 Modeling Uncertainty

When we apply this transformation, we are interested in the *absolute perforation noise* — that is, the absolute value of the difference between the final value of `sum` that the original loop produces and the final value of `sum` that the transformed loop produces. Obviously, the absolute perforation noise will depend on the values of `f(i)`. The standard approach to reasoning about program transformations considers the worst case. In our example, this worst-case reasoning would consider the largest possible absolute perforation noise, then apply the transformation only if this worst case absolute perforation noise is acceptable.

We choose to take a different approach because, while we do not have complete information about the values of `f(i)`, we do believe that this worst case is very unlikely to occur in practice. We prefer to analyze the transformation from a perspective that enables a more accurate analysis of the effect of the transformation. We therefore formally characterize our uncertainty about the values of `f(i)` by modeling these values as random variables. The distribution of these random variables captures our uncertainty about their values.

We then reason about the effect of the transformation by studying the resulting absolute perforation noise, itself a random variable, whose distribution is determined by the original program, the applied transformation, and the distributions of the random variables that model the values that the computation manipulates. An analysis of the resulting properties of the distribution of the absolute perforation noise determines whether the transformation is acceptable or not, given our uncertainty about the input.

1.1.2 Independent Random Variables

We first consider the case when we believe that the value of each `f(i)` is independent of the values of all other `f(i)` and that `f(i)` always takes on a value in the interval $[a, b]$. We model this situation by assuming that the `f(i)` are independent, identically distributed (i.i.d.) samples chosen from the uniform distribution $U(a, b)$ on the interval $[a, b]$ with expected value $\mu = \frac{1}{2}(a + b)$ and variance $\sigma^2 = \frac{1}{12}(b - a)^2$.

We use the following notation. S_O is the final value of `sum` in the original program, S_P is the final value of `sum` in the perforated program, and X_i are the n random variables that model the values `f(i)`. To simplify the analysis, we require that n is even. Because all of the X_i come from the same distribution, the expected value $\mathbb{E}(S_O)$ of S_O equals the expected value $\mathbb{E}(S_P)$ of S_P. Conceptually, S_P is therefore an unbiased approximation to S_O.

The *perforation noise* D is the difference $D = S_O - S_P$ between the results that the original and perforated programs produce. Note that

$$S_O = \sum_{0 \le i < \frac{n}{2}} (X_{2i} + X_{2i+1}) \tag{1}$$

and

$$S_P = \sum_{0 \le i < \frac{n}{2}} (X_{2i} + X_{2i}) \tag{2}$$

so

$$D = \sum_{0 \le i < \frac{n}{2}} (X_{2i+1} - X_{2i}) \tag{3}$$

and D is therefore a random variable with expected value $\mathbb{E}(D) = 0$ and variance $\mathrm{var}(D) = n\sigma^2$.

The perforation noise D may be either positive or negative. We may therefore be interested in the magnitude of the difference between the results that the original and transformed loops compute. We call the magnitude $|D|$ the *absolute perforation noise*. To reason about $|D|$, we write D as

$$D = \sum_{0 \le i < \frac{n}{2}} X_{2i+1} - \sum_{0 \le i < \frac{n}{2}} X_{2i} \tag{4}$$

and note that, as n increases, the distribution of the sum of uniformly distributed random variables rapidly approaches a normal distribution. We therefore approximate D as the difference between two normally distributed random variables, each with expected value $\frac{n}{2}\mu = \frac{n}{4}(a+b)$ and variance $\frac{n}{2}\sigma^2 = \frac{n}{24}(b-a)^2$. So D is normally distributed with expected value 0 and variance $n\sigma^2$, and $|D|$ is half-normally distributed with expected value $\mathbb{E}(|D|) = \sigma\sqrt{\frac{2n}{\pi}}$ and variance $\mathrm{var}(|D|) = n\sigma^2(1 - \frac{2}{\pi})$.

We note that over 99.7% of values drawn from a normal distribution are within three standard deviations of the mean. The mean of the perforation noise D is 0, the standard deviation is $(b-a)\sqrt{\frac{n}{12}}$, and so $\mathbb{P}(|D| \ge (b-a)\sqrt{0.75n}) \le 0.003$.

Alternatively, we can use Hoeffding's inequality (see Section 2) to bound the probability of observing large absolute perforation noise. Specifically,

$$\mathbb{P}\left(|D| \ge \sqrt{\frac{n(b-a)^2}{2} \ln \frac{2}{\epsilon}}\right) \le \epsilon, \quad \epsilon \in (0, 1). \tag{5}$$

So, (setting $\epsilon = 0.01$), with probability greater than 0.99, the absolute perforation noise $|D|$ is less than $(b-a)\sqrt{2.649n}$.

A comparison with the traditional worst-case analysis is instructive. Specifically, the (we believe extremely unlikely) worst-case absolute perforation noise is $\frac{n}{2}(b-a)$. Note that reasoning about the computation from the probabilistic perspective as opposed to the traditional worst-case perspective produces a factor of $\sqrt{n}$ asymptotic improvement in the bound on the absolute perforation noise.

1.1.3 Correlations via Overlapping Sums

We next consider a case in which we believe that the values of `f(i)` are correlated. This can happen, for example, if the `f(i)` come from overlapping sums of values from a given domain (this happens, for example, for some of the computations in the x264 video encoder application [4]). We model this situation by starting with a sequence Y_j of independent, identically distributed (i.i.d.) samples chosen from the uniform distribution $U(a, b)$ on the interval $[a, b]$ with expected value $\mu = \frac{1}{2}(a + b)$ and variance $\sigma^2 = \frac{1}{12}(b - a)^2$. For a *window size* w the sequence X_i is given by

$$X_i = \sum_{0 \le j < w} Y_{j+i}. \tag{6}$$

In this case each X_i is a random variable with expected value $\frac{w}{2}(a+b)$ and variance $\frac{w}{12}(b-a)^2$. However, the difference $X_{i+1} - X_i$ is precisely $Y_{i+w} - Y_i$, which has variance $\frac{1}{6}(b-a)^2$ (as opposed to variance $\frac{w}{6}(b-a)^2$ for the difference between independent X_i with the same variance). The perforation noise is:

$$D = \sum_{0 \le i < \frac{n}{2}} (Y_{2i+w} - Y_{2i}). \tag{7}$$

For simplicity we assume that $w \ge 3$ and w is odd. Then the remaining analysis is identical to the analysis resulting from Equation 3 and provides probabilistic bounds of the same form that are a factor w tighter. Note that the correlation between adjacent X_i enables the analysis to obtain tighter probabilistic bounds relative to the variance of the random variables X_i used to model the values of the summands.

1.1.4 Truncation Perforation for Overlapping Sums

It is instructive to compare the interleaved perforation transformation in Figure 2 (which skips every other loop iteration) with the truncation perforation transformation in Figure 3 (which skips the last half of the iterations). If the values `f(i)` are independent, the analysis of the truncated computation is identical to the analysis in Section 1.1.2 and the resulting probabilistic bounds are identical.

If, however, the values `f(i)` are correlated as in Section 1.1.3, the skipped values from iterations `n/2` through `n-1` are (if `n` is significantly larger than w) largely uncorrelated with the values from iterations `0` through `n/2-1`. The perforation noise D of the truncated computation therefore has a factor of w larger variance, with correspondingly looser probabilistic bounds.

```
double sum = 0.0;
for (int i = 0; i < n/2; i++) {
  sum += f(i);
}
sum = sum * 2;
```

Figure 3. Loop After Truncation Perforation

As this example illustrates, one reason to prefer interleaved perforation is that it can deliver smaller perforation noise than truncated perforation when values from nearby loop iterations are more strongly correlated than values from distant loop iterations.

1.1.5 Correlations via Random Walks

We next consider another case in which the values of `f(i)` are correlated. In this case we believe that the `f(i)` are generated by a Markov process in which the difference between adjacent values is a normally distributed random variable with mean 0 and

variance σ^2. We model the value of `f(i)` with a random variable X_i, where $X_{i+1} = X_i + Y_i$, $0 \leq i$. Here the Y_i are independent random variables chosen from a normal distribution with mean 0 and variance σ^2. For simplicity we let $X_0 = 0$. In this case the perforation noise D is:

$$D = \sum_{0 \leq i < \frac{n}{2}} (X_{2i+1} - X_{2i}) = \sum_{0 \leq < \frac{n}{2}} Y_{2i}. \quad (8)$$

So D is a normally distributed random variable with mean 0 and variance $\frac{n}{2}\sigma^2$. As in Section 1.1.2, the absolute perforation noise $|D|$ has a half-normal distribution. The analysis of the mean, variance, and probability of observing large $|D|$ is similar to the corresponding analysis in Section 1.1.2, but with the variance of D a factor of 2 smaller ($\frac{n}{2}\sigma^2$ as opposed to $n\sigma^2$).

The mean of the perforation noise D is 0, the standard deviation is $\sigma\sqrt{\frac{n}{2}}$, so (because over 99.7% of the values drawn from a normal distribution are within three standard deviations of the mean) $\mathbb{P}(|D| \geq \sqrt{\frac{9n}{2}}) \leq 0.003$.

1.1.6 Truncation Perforation for Random Walks

We next compare the effect of interleaved perforation with the effect of truncation perforation (see Section 1.1.4) for the random walk correlated values of `f(i)` from Section 1.1.5. With truncation perforation we can write the perforation noise D as:

$$D = \sum_{0 \leq i < \frac{n}{2}} X_{\frac{n}{2}+i} - X_i. \quad (9)$$

Applying the equality $X_i = \sum_{0 \leq j < i} Y_j$, we can rewrite D as:

$$D = \sum_{0 \leq i < \frac{n}{2}} (i+1)Y_i + \sum_{\frac{n}{2} \leq i < n-1} (n-(i+1)Y_i. \quad (10)$$

We can rearrange terms to rewrite D as:

$$D = \sum_{0 \leq i < \frac{n}{2}} \sum_{0 \leq j < \frac{n}{2}} \text{if } (i < j) \ Y_i \text{ else } Y_{n-(i+2)}. \quad (11)$$

Because the Y_i are i.i.d random variables chosen from a normal distribution with mean 0 and variance σ^2, the variance of D is $(\frac{n}{2})^2\sigma^2$. Compare this variance with the variance of D from the transformed program with interleaved perforation, which is $\frac{n}{2}\sigma^2$. Once again, we see how interleaved perforation can, when values from nearby iterations are more strongly correlated than values from distant iterations, deliver computations with significantly smaller perforation noise variance and correspondingly tighter probabilistic bounds than truncation perforation.

1.1.7 Arbitrary Linear Functions

This example presented a simple sum computation. We note, however, that the probabilistic analysis (as well as related probabilistic analyses presented in Section 3) generalizes to include computations that compute arbitrary linear functions of the input variables (as well as any abstracted expressions, see Section 5). And as the analyses in Section 3 show, it also possible to generalize the analysis to handle computations that include non-linear operations such as minimum and division.

1.2 Loop Perforation

In this technical report we consider the *loop perforation* transformation [9, 11]. Loop perforation transforms loops to execute only a subset of the 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 program requires to produce a result while preserving acceptable accuracy [9].

The empirical results show that the effect of loop perforation is different for different loops. In some cases loop perforation produces small or even negligible changes in the result. In others, loop perforation can cause large changes in the result or even cause the program to crash. One of the contributions of this technical report is the identification of specific computational patterns that interact well with loop perforation. Included in this identification is a precise characterization of the effect of loop perforation on the result that the pattern produces. This characterization provides the foundation for the reasoning required to justify the principled application of loop perforation to specific loops.

1.3 Modeling and Analysis

We quantify the effect of the perforation by defining the *perforation noise* — the difference between the result that the original computation produces and the result that the perforated computation produces. We model the computation as operating on sets of random numbers selected from different probability distributions. We identify specific computational patterns that interact well with the loop perforation transformation. For each pattern our analysis produces simple mathematical expressions that characterize the expected value and variance of the perforation noise. Our analysis also produces expressions that characterize the probability of observing large absolute perforation noise. These expressions are parameterized by values such as the number of iterations that the original loop performs, the percentage of iterations that the transformed program does not execute, and properties of the probability distributions such as their mean and variance.

We propose several mechanisms for obtaining concrete values for the parameters of the mathematical expressions. For probability distribution parameters (such as the mean and variance), we propose the use of techniques that fit the distributions to data observed in representative executions. From a Bayesian perspective, we start out with a belief about the class of distributions (normal, uniform, etc.) that accurately model the values that the computation manipulates. We then update this belief in light of observations from representative executions to choose a specific distribution. We may then simply use this distribution for future executions, or we may choose to continue to observe values and update our distributions accordingly. Periodic sampling techniques such as dynamic feedback [6] may be able to drive down the dynamic overhead of this approach.

We note that our usage scenario differs from many usage scenarios in that we may be interested in not the most accurate model, but a model that conservatively overestimates quantities such as the absolute perforation noise. The application of such conservative techniques may cause the transformation system to miss transformation opportunities, but may also reduce the chances that a modeling inaccuracy will cause the transformed program to produce unacceptably inaccurate results.

For other values (such as the number of loop iterations), we may be able to determine the value before the program executes through traditional program analysis. We may also generate code that dynamically tests the value to see if the transformed computation will produce acceptable results with that value. If so, the generated code branches to execute the transformed computation. If not, it executes the original computation.

1.4 Empirical Evaluation

In general, the validity of the expressions that characterize the perforation noise may depend on how we model the values on which the computation operates (for example, we sometimes model the values as independent and identically distributed). We evaluate the accuracy of the model by generating instrumented programs to collect execution data. These programs record the values that the computation operates with and the results from both the original and

```
double sum = 0.0;
double _sum = 0.0;
double _t;
log("start perforated loop %d", n);
for (int i = 0; i < n; i++) {
  _t = f(i);
  log("expression %d %lf", i, _t);
  sum += t;
  if (i % 2) _sum += t;
}
log("end perforated loop %lf %lf",
  sum, _sum);
```

Figure 4. Instrumented Program

transformed computations. Figure 4 presents a (high-level version) of the instrumented program from our example.

We run the instrumented programs on representative inputs and inspect the collected information. We use the recorded input statistics to obtain concrete values for the parameters of the mathematical expressions that the analysis produces. We then either perform statistical tests to evaluate how accurately our distributions model the numbers on which the computation operates or simply evaluate how well the model predicts the perforation noise observations from actual executions of the original and transformed programs. Note that models that conservatively overestimate the observed perforation noise may be acceptable in most usage scenarios.

1.5 Probabilistic Accuracy Specifications

To apply transformations that may change the result that the program produces, we must understand what kinds of changes are acceptable. In general, we anticipate that computations for which rich transformations are appropriate may be embedded inside larger computations for which symbolic reasoning about the effect of the rich transformation is infeasible.

We therefore envision the use of *probabilistic acceptability specifications* to help the transformation system determine which transformation to apply. These specifications are designed to allow developers to precisely characterize their uncertainty about the values on which the computation operates. They are also designed to allow developers to express their probabilistic accuracy requirements for the transformed computation. Each probabilistic acceptability specification applies to a specific computation. These specifications include the following components:

- **Input Specifications:** For each input to the computation, a specification of an appropriate probability distribution to model the values of the input. This specification can be symbolic (for example, a normal distribution with specified mean and variance or a uniform distribution with specified upper and lower bounds) or empirical (for example, a histogram that approximates the distribution of the input values).

  We consider several ways to obtain these distributions. First, the developer may provide the full symbolic distribution, including distribution parameters such as mean and variance. Second, the developer may provide the general class of the distribution (such as normal or uniform), but rely on the system to use empirically collected data from representative executions to fit the parameters of the distribution to the collected data. Third, the system may use statistical reasoning over the collected data to choose both the general class of the distribution and to fit the parameters to the chosen data. Finally, the system may simply use an automatically computed histogram of the collected data values as its probability distribution.

- **Expression Specifications:** In some cases it is appropriate for the analysis to model the values of expressions as random variables. Examples include complex expressions or function calls for which the source code is not available in analyzable form. Each expression specification identifies an expression and an appropriate probability distribution to model the values of this expression. As for input specifications, expression specifications can be symbolic or empirical and specified by the developer or derived by observing values in representative executions.

- **Output Specifications:** For each output of the computation, a specification of acceptable variation from the value that the untransformed computation produces. There are two kinds of output specifications:

  - **Moment-Based:** Moment-based specifications identify acceptable moments of the perforation noise for that output. For example, the developer may specify that the mean and/or variance of the perforation noise should be below some value.

  - **Bound-Based:** Bound-based specifications identify an acceptable probability that the perforation noise will exceed a given bound. For example, the developer may specify that, with probability at least 95%, the perforation noise should be less than 1. We anticipate the use of both absolute and relative measures (absolute measures are simply the difference between the values that the original and transformed computations produce; relative measures divide this difference by the magnitude of the result of the untransformed computation).

This approach promotes the use of a programming model in which the developer specifies a maximally accurate version of the computation, then relies on rich transformations to deliver a version with an appropriate balance between accuracy and performance.

1.6 Usage Scenarios

The simplest usage scenario involves a single computation with a given acceptability specification that the transformation system transforms to maximize performance subject to satisfying the acceptability specification. A more complex usage scenario involves multiple optimizable computations embedded within a larger computation. Here the transformations can induce a complex performance versus accuracy trade-off space. The transformation system can use executions on representative inputs to explore this space and discover Pareto-optimal points within this space. Depending on the specific usage scenario, the application may choose one of these versions or even switch dynamically between versions as necessary to preserve computing goals in the face of environments with fluctuating characteristics [8, 9].

1.7 Contributions

This technical report makes the following contributions:

- **New Paradigm:** It presents a new paradigm for justifying program transformations. Instead of using discrete logic to justify transformations that preserve the exact semantics of the program, it uses statistical and probabilistic reasoning to justify transformations that may, within statistical and probabilistic bounds, change the result that the program produces.

  In this approach, we model the values which the computation manipulates as random variables. The distributions of these random variables characterize our uncertainty about their values. We use these probability distributions (in combination with the original program and the applied transformation) to reason about the effect of the transformation on the values that

the computation produces. Specifically, we model differences between the results that the original and transformed computations produce as random variables. The analysis extracts properties of these random variables such as their mean, variance, and probabilistic bounds on their magnitude. These properties determine whether or not the transformation is acceptable.

- **Patterns:** It presents a set of computational patterns with transformations that can be analyzed using probabilistic analyses. Each pattern consists of the description of the computation that it performs, the transformation that changes the result and the probabilistic analysis. It uses loop perforation as the transformation strategy for all of the computations.
- **Analysis:** It presents a set of probabilistic analyses for identified patterns. Each analysis provides expressions for the expected value and variance of the absolute perforation noise. We also provide bounds that characterize the probability of observing large probability noise and a comparison with a standard deterministic worst-case analysis of the effect of loop perforation.
- **Specification:** It introduces the concept of probabilistic acceptability specifications, which specify probability distributions for modelling the values of inputs and expressions within blocks of code and probabilistic constraints on the absolute difference between the results that original and transformed blocks of code produce.
- **Evaluation:** It presents a statistical evaluation of our analyses on a set of transformed computational patterns from the Parsec benchmark suite [4]. For each computation, the technical report presents a comparison between the predicted and observed perforation noise. We also compare the results of the probabilistic analyses with standard deterministic worst-case analyses.

The remainder of the technical report is structured as follows. Section 2 briefly reviews loop perforation and some basic probability theory results. Section 3 presents the code patterns and probabilistic analysis of the patterns. Section 3 also presents a more conservative worst-case analysis of the perforated patterns. Section 4 presents the results of the probabilistic analyses on a set of representative computations from complex programs. Section 5 presents our probabilistic specification approach. Section 6 discusses ways to automate the approach. Section 7 discusses related work.

2. Preliminaries

In this section, we review loop perforation and some elementary probability theory that we use in our analyses.

2.1 Loop Perforation

Loop perforation is a program transformation that modifies the number of iterations that a `for` loop performs. It modifies the loop termination condition, the induction variable initialization expression, or the induction variable increment or decrement expression to decrease the amount of work performed in the loop.

If the original loop executes for n iterations, the perforated loop will execute m, $m < n$ iterations. We take the number of loop iterations as a representative measure of the work performed in the loop. The developer may specify the expected amount of skipped work using a *perforation rate*, the expected percentage of loop iterations that should be skipped. Given a perforation rate r, the expected number of iterations of the perforated loop is $\bar{m} = n(1-r)$. Note that in a specific perforated computation the number of executed iterations, m, is only approximately equal to $\bar{m}$, because m must be an integer. Typically, $m = \lfloor \bar{m} \rfloor$ or $m = \lceil \bar{m} \rceil$. The actual number of iterations in general depends on the loop perforation implementation.

Loop perforation can reduce the amount of work in a loop in various ways. A loop perforation *strategy* can be represented by a $n \times 1$ *perforation vector* P where each coordinate corresponds to a single loop iteration. The number of non-zero coordinates of vector P is equal to m. The all-ones perforation vector $A = (1, \ldots, 1)'$ corresponds to the unperforated computation.

Patterns may assign additional semantics to the non-zero coordinates of the vector. For example, the extrapolated sum pattern (Section 3.1) multiplies the summand calculated in i-th iteration of the perforated version of the code by the extrapolation coefficient stored in the i-th coordinate of P.

Perforation implementation. Loop perforation can be applied to any loop that has an induction variable. Loop perforation can successfully handle loops with arbitrary initial induction variable values, arbitrary induction variable increments or decrements, and arbitrary loop termination conditions. Without loss of generality, we will study loops whose inductive variable takes an initial value 1, whose inductive operation at each step is incrementation, and whose final condition tests whether the value is less than or equal to n.

Conceptually, the implementation of loop perforation calculates the vector P before the loop executes based on the number of original iterations and the perforation rate. It executes only those loop iterations for which the coordinate P_i has a non-zero value.

```
P = calc_perforation_vector(n, r)
for (int i = 1; i <= n; i++) {
   if ( !P[i] ) continue;
   ...
}
```

In most cases we will not need to explicit create the P vector. In this section we will describe a few common perforation strategies and their more specialized implementations. These strategies include interleaving, truncation and random choice.

Large interleaving perforation. Large interleaving perforation prescribes the execution of every k-th iteration, where $k = \lfloor \frac{n}{m} \rfloor$. The standard large interleaving perforation pattern is:

```
for (int i = 1; i <= n; i+=k) {...}
```

The corresponding perforation vector without extrapolation has coordinates $P_{ki+1} = 1$, where $i \in \{0, ...m-1\}$, and $P_{ki+j} = 0$ for $j \in \{2, ...k\}$. Note that we start iterating from the first element of the vector as a matter of convention. It is trivial to extend the perforation strategy to start from some other index.

Small interleaving perforation. Small interleaving perforation skips every k-th iteration, where $k = \lfloor \frac{n}{n-m} \rfloor$. The standard small interleaving perforation pattern is:

```
for (int i = 1; i <= n; i+= i%k? 1:2 ) {...}
```

The corresponding perforation vector has coordinates $P_{k(i+1)} = 0$, $P_{ki+j} = 1$, where $i \in \{0, ...m-1\}$ and $j \in 1, \ldots, k-1$ otherwise.

Truncation perforation. Truncation perforation skips contiguous ranges of loop iterations, usually $n - m$ iterations at the end or at the beginning of the loop. A standard truncation perforation pattern is (this pattern skips iterations from the end of the loop):

```
for (int i = 1; i <= m; i++) {...}
```

The corresponding perforation vector has coordinates $P_i = 1$, for $i \leq m$, and $P_i = 0$ otherwise. It is straightforward to extend this strategy for initial loop induction variable values other than 1.

Randomized perforation. Randomized perforation selects a random subset of m elements from the entire set of n elements. This can be done efficiently without explicitly creating the P vector.

2.2 Perforation Noise

We characterize the difference between the original and perforated computations using the *perforation noise*. For an input vector $X = (X_1, \ldots, X_n)$ and perforation vectors A and P, let calc be a perforatable computation that given input vector X and a perforation strategy P calculates an output. We define the (signed) perforation noise by

$$D \equiv \mathsf{calc}(X, P) - \mathsf{calc}(X, A). \tag{12}$$

Then the absolute perforation noise is the absolute value $|D|$.

Specially, if the computation sums its inputs, we can model the computation by taking the dot product of the perforation vector with the input vector, $\mathsf{calc}(X, P) = P'X$. For the original computation the dot product of the two vectors will yield

$$A'X = \sum_{i=1}^{n} A_i X_i = \sum_{i=1}^{n} X_i \tag{13}$$

For the perforated sum S_m we can derive

$$P'X = \sum_{i=1}^{n} P_i X_i = \sum_{i:P_i \neq 0}^{n} P_i X_i \tag{14}$$

Then the absolute perforation noise $|D|$ is:

$$|D| = \left| \sum_{i:P_i \neq 0}^{n} P_i X_i - \sum_{i=1}^{n} X_i \right| \tag{15}$$

2.3 Useful Inequalities

In this section we describe a few useful inequalities that we will use during the pattern analyses.

Markov's inequality. For a random variable Y with finite mean, Markov's inequality provides an upper bound on the probability of observing its absolute value exceed a. In particular,

$$\mathbb{P}\Big(|Y| \geq a\Big) \leq \frac{\mathbb{E}\big(|Y|\big)}{a}. \tag{16}$$

This inequality is non-trivial when $a > \mathbb{E}(|Y|)$.

Chebyshev's inequality. For a random variable Y with finite mean and variance, a *two-sided Chebyshev's inequality* gives an upper bound on the probability of observing an absolute difference between Y and its mean larger than some value a. In particular,

$$\mathbb{P}\Big(|Y - \mathbb{E}(Y)| \geq a\Big) \leq \frac{\mathrm{var}(Y)}{a^2}. \tag{17}$$

We can restate Chebyshev's inequality in the following way. With probability at least $1 - \epsilon$, the quantity $|Y - \mathbb{E}(Y)|$ is bounded by

$$\sqrt{\frac{\mathrm{var}(Y)}{\epsilon}}. \tag{18}$$

One-sided Chebyshev's inequality provides the bound on Y being greater than $\mathbb{E}Y$. In particular, we have

$$\mathbb{P}\Big(Y - \mathbb{E}(Y) \geq a\Big) \leq \frac{\mathrm{var}(Y)}{\mathrm{var}(Y) + a^2}. \tag{19}$$

Alternatively, we can say that with probability at least $1 - \epsilon$, the difference $Y - \mathbb{E}(Y)$ is bounded by

$$\sqrt{\mathrm{var}(Y)\Big(\frac{1}{\epsilon} - 1\Big)}. \tag{20}$$

Hoeffding's inequality. For a sum of random variables $S_n = X_1 + \ldots + X_n$ where all terms X_i are independent and almost surely bounded, i.e. $P(X_i \in [a_i, b_i]) = 1$, *a two-sided Hoeffding's inequality* gives an upper bound on the absolute difference between S_n and its mean larger than some constant t. In particular,

$$\mathbb{P}\Big(\Big|S_n - \mathbb{E}S_n\Big| \geq t\Big) \leq 2\exp\Big(-\frac{2t^2}{\sum_{i=1}^{n}(b_i - a_i)^2}\Big). \tag{21}$$

Because of additional assumptions on the input random variables, Hoeffding's inequality often provides tighter bounds than Chebyshev's or Markov's inequality.

We can restate the Hoeffding's inequality in the following way. With probability at least $1 - \epsilon$, the absolute difference $|S_n - \mathbb{E}S_n|$ is bounded by

$$\sqrt{\frac{1}{2}\ln\frac{2}{\epsilon} \cdot \sum_{i=1}^{n}\Big(b_i - a_i\Big)^2}. \tag{22}$$

One-sided Hoeffding's inequality provides the bound on the sum S_n being greater than $\mathbb{E}S_n$. In particular, we have

$$\mathbb{P}\Big(S_n - \mathbb{E}S_n \geq t\Big) \leq \exp\Big(-\frac{2t^2}{\sum_{i=1}^{n}(b_i - a_i)^2}\Big). \tag{23}$$

Alternatively, we can say that with probability at least $1 - \epsilon$, the difference $S_n - \mathbb{E}S_n$ is bounded by

$$\sqrt{\frac{1}{2}\ln\frac{1}{\epsilon} \cdot \sum_{i=1}^{n}\Big(b_i - a_i\Big)^2}. \tag{24}$$

3. Patterns and Analyses

For each pattern we present an example of the original and the transformed code. For simplicity, in these examples we use a large interleaving perforation and assume an even number of iterations n.

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 about the input distributions or perforation strategy.

3.1 The Extrapolated Sum Pattern

Original code	Transformed Code

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

Figure 5. Extrapolated Sum Pattern; original and perforated code

Figure 5 presents an example of the original and the perforated code for the extrapolated sum pattern. We begin by first presenting 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 from a random walk.

Assumptions. To derive the bounds on the perforation noise, we will assume only that the terms of the sum have finite mean and covariance.

Analysis. For $i = 1, \ldots, n$, let $X_i = \texttt{f}(i)$ be the i-th term of the summation. We will model our *uncertainty* about the values X_i by treating $X = (X_1, \ldots, X_n)'$ as a vector of n random variables with mean vector

$$M = (\mu_1, \ldots, \mu_n)' \tag{25}$$

and covariance matrix Σ with elements

$$(\Sigma)_{ij} = \text{cov}(X_i, X_j). \tag{26}$$

Let A be the all-ones vector, so that $A'X$ denotes the sum $\sum_{i=1}^{n} X_i$, and let P be a perforation vector with m non-zero entries. Then $P'X$ denotes the extrapolated perforated computation. The signed perforation noise is

$$D \equiv P'X - A'X = (P - A)' X. \tag{27}$$

In the general case, the expected value of the perforation noise is

$$\mathbb{E}(D) = \sum_{i=1}^{n} \mu_i \cdot (P_i - 1), \tag{28}$$

and its variance is given by

$$\text{var}(D) = (P - A)' \Sigma (P - A) \tag{29}$$

$$= \sum_{i,j} (P_i - 1)\,(P_j - 1)\,\Sigma_{i,j}. \tag{30}$$

To avoid systematic bias, we can try to choose P so that $\mathbb{E}(D) = 0$. For a known mean vector, M, this is indeed possible.[1] In particular, choosing P such that

$$\sum_{i=1}^{n} \mu_i P_i = \sum_{i=1}^{n} \mu_i \tag{31}$$

removes a bias from the extrapolation.

As a special case, if all input variables have the same expected value, i.e., if $\mu = \mu_1 = \ldots = \mu_n$, then it follows that P will lead to an unbiased estimate if and only if

$$\sum_{i=1}^{n} P_i = n. \tag{32}$$

One common extrapolation strategy is to assign the same value to every non-zero coordinate, in which case we would have $P_i = \frac{n}{m}$ for the non-zero coordinates i.

Under the assumption that P satisfies $\mathbb{E}(D) = 0$, we can use Chebyshev's inequality and the variance of D to bound the absolute perforation noise by

$$\mathbb{P}(|D| > a) \leq \frac{\text{var}(D)}{a^2}. \tag{33}$$

This bound will be conservative in practice; additional knowledge (say, bounds on the range of each X_i and independence) can be used to derive tighter bounds, e.g., by using Hoeffding's inequality. Alternatively, from those P satisfying $\mathbb{E}(D) = 0$, we may choose the one that minimizes the variance, and therefore minimizes the bound.

We will now study a number of special cases where additional assumptions will enable us to better understand the effect of perforation.

[1] Note that the mean vector and covariance matrix may themselves be uncertain and so it may not be possible to choose a P for which $\mathbb{E}(D) = 0$. Distributional assumptions on the mean and covariance would allow one to, e.g., compute the conditional expectation of the unperforated sum given the perforated computations, producing an estimate that minimizes a mean squared error criterion. We leave this approach to future work.

3.1.1 Independent Variables

Assumptions. We will assume that the elements $X_i = \texttt{f}(i)$ of the summation are i.i.d. copies of a random variable with finite mean μ and variance σ^2. (In order to derive a mean and variance for the absolute perforation noise, we will consider the additional assumption that X is normally distributed.)

Analysis. Because X is a vector of i.i.d elements, it follows that $\Sigma = \text{diag}_i(\sigma^2)$, where $\text{diag}_i(e_i)$ denotes the diagonal matrix whose (i, i) entry is e_i. Moreover, $\mathbb{E}(X_i) = \mu$, and so, from (32), we have

$$\mathbb{E}(D) = 0, \tag{34}$$

for any perforation P such that $\sum_i P = n$. From Eq. 30 and the observation that $\Sigma = \text{diag}_i(\sigma^2)$, it follows that

$$\text{var}(D) = \sigma^2 \sum_i (1 - P_i)^2\,. \tag{35}$$

It is straightforward to show that this value is minimized by any perforation P with $n - m$ zeros and the remaining entries taking the value $\frac{n}{m}$. In this case, the variance takes the value

$$\frac{\sigma^2\, n\,(n - m)}{m}. \tag{36}$$

We can immediately bound the probability of observing large absolute perforation noise using Chebyshev's inequality (Eq. 33). In particular, with probability at least $1 - \epsilon$,

$$|D| \leq \sqrt{\frac{\sigma^2\, n\,(n - m)}{m\,\epsilon}} \tag{37}$$

If each X_i is bounded, falling in the range $[a, b]$, we can apply Hoeffding's inequality (Eq. 22) to bound the absolute perforation noise $|D|$. Let $X_i' = (P_i - 1)X_i$, and note that the variables X_i' are still mutually independent. The range of X_i' is $[a_i, b_i]$, which is equal to $[(P_i - 1)a, (P_i - 1)b]$. Then the sum

$$\sum_{i=1}^{n} \Big(b_i - a_i\Big)^2 = (b - a)^2 \Big(m \Big(\frac{n}{m} - 1\Big)^2 + (n - m)\Big) \tag{38}$$

$$= (b - a)^2 \frac{n\,(n - m)}{m}. \tag{39}$$

It follows that, with probability at least $1 - \epsilon$,

$$|D| < (b - a) \sqrt{\frac{n\,(n - m)}{2m} \ln \frac{2}{\epsilon}}. \tag{40}$$

We can get potentially tighter bounds if we make further assumptions. Let D^* denote the perforation noise in the case where summand X is normally distributed with mean μ and variance σ^2. Then D^* is itself normally distributed with mean and variance given by (34) and (35), and thus the absolute perforation noise $|D^*|$ has a half-normal distribution with mean

$$\mathbb{E}\Big(|D^*|\Big) = \sqrt{\frac{2}{\pi} \text{var}(D^*)} \tag{41}$$

$$= \sigma \sqrt{\frac{2n(n - m)}{\pi m}} \tag{42}$$

and variance

$$\text{var}(|D^*|) = \Big(1 - \frac{2}{\pi}\Big) \text{var}(D^*). \tag{43}$$

3.1.2 Random Walk

Assumptions. We will assume that the sequence X of random variables is a random walk with independent increments. Specifically, we assume 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 $\mathbb{E}(X_i) = \mu$. (To see this, note that $\mathbb{E}(X_i) = \mathbb{E}(X_{i-1}) + \mathbb{E}(\delta_{i-1})$ and $\mathbb{E}(X_0) = \mu$.) As a consequence, for any perforation vector that satisfies (32), we have that $\mathbb{E}(D) = 0$.

For $i < j$, the covariance between X_i and X_j satisfies

$$\begin{aligned}\operatorname{cov}(X_i, X_j) &= \mathbb{E}\Big((X_i - \mu)(X_j - \mu)\Big) && (44)\\ &= \mathbb{E}\Big((X_i - \mu)\mathbb{E}(X_j - \mu | X_i)\Big) && (45)\\ &= \mathbb{E}\Big((X_i - \mu)^2\Big) && (46)\\ &= \operatorname{var}(X_i) && (47)\\ &= i\sigma^2. && (48)\end{aligned}$$

Therefore, the covariance matrix Σ has entries

$$(\Sigma)_{ij} = \sigma^2 \min\{i, j\}, \tag{49}$$

and the variance of the perforation noise satisfies

$$\operatorname{var}(D) = \sigma^2 \sum_{i,j} (1 - P_i)\,(1 - P_j) \min\{i, j\}. \tag{50}$$

We may choose a perforation strategy P by minimizing this variance (and thus minimizing the Chebyshev's bound on the absolute perforation noise). For example, when $P_i = 2$ for odd i and 0 otherwise, we have that

$$\operatorname{var}(D) = \frac{n}{2}\sigma^2. \tag{51}$$

3.2 Mean Pattern

Original code	Transformed Code

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

Figure 6. Mean Pattern; original and perforated code

We present an example of the original and the perforated code for the mean pattern in Figure 6. Note that we can extend the analysis for the extrapolated sum pattern (Section 3.1) because the result of the mean computation is equal to the result of the sum computation divided by n. We denote the output produced by the original computation as $\frac{1}{n}A'X$, and the output produced by the perforated computation as $\frac{1}{n}P'X$.

We denote the perforation noise of the sum as D_S. The perforation noise of the mean D in the general case with correlated variables is

$$D \equiv \frac{1}{n}\big(P'X - A'X\big) \tag{52}$$

$$D = \frac{1}{n}D_S \tag{53}$$

By the linearity of expectation, the perforation noise has expectation

$$\mathbb{E}(D) = \frac{1}{n}\mathbb{E}(D_S) \tag{54}$$

and variance

$$\operatorname{var}(D) = \frac{1}{n^2}\operatorname{var}(D_S). \tag{55}$$

For perforation vectors such that $\mathbb{E}D = 0$, we have that

$$\mathbb{P}(|D| > a) \leq \frac{\operatorname{var}(D_S)}{n^2 a^2}. \tag{56}$$

Therefore, if $\frac{1}{n}\operatorname{var}(D_S) \to 0$ as $n \to \infty$, then we can expect the perforated computation to approximate the original computation to any desired accuracy with high probability for large enough n.

3.2.1 Independent Variables

Assumptions. As we did in the sum case in Section 3.1.1, we will consider the special case where the terms $X_i = \mathtt{f}(i)$ are i.i.d. copies of a random variable with finite mean μ and variance σ^2. (Again, in order to derive the mean and variance for the absolute perforation noise, we will make the additional assumption that the X_i are normally distributed.)

Analysis. We derive the results in this section using the results from Section 3.1.1.

By construction, the expected perforation noise satisfies

$$\mathbb{E}(D) = 0. \tag{57}$$

The variance of the perforation noise is

$$\operatorname{var}(D) = \Big(\frac{1}{m} - \frac{1}{n}\Big)\sigma^2. \tag{58}$$

This identity can then be used to derive Chebyshev bounds via (56).

We can obtain tighter bounds on the perforation noise if we make the additional assumption that the summands X_i are normally distributed. In this case the perforation noise D is also normally distributed, and so $|D|$ has a half-normal distribution with mean

$$\mathbb{E}(|D|) = \sigma\sqrt{\frac{2}{\pi}\left(\frac{1}{m} - \frac{1}{n}\right)} \tag{59}$$

and variance

$$\operatorname{var}(|D|) = \sigma^2\left(1 - \frac{2}{\pi}\right)\left(\frac{1}{m} - \frac{1}{n}\right). \tag{60}$$

Alternatively, if we constrain the variables X_i to lie in the interval $[a, b]$, we can use Hoeffding's inequality (Eq. 22) to bound the chance of observing a large perforation noise. With probability $1 - \epsilon$, the absolute perforation noise is bounded by

$$|D| < (b - a)\sqrt{\frac{1}{2}\Big(\frac{1}{m} - \frac{1}{n}\Big)\ln\frac{2}{\epsilon}}. \tag{61}$$

3.3 Argmin-Sum Pattern

We present an example of the original and transformed code for the Argmin-Sum computation pattern in Figure 7.

Original code	Transformed Code

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

Figure 7. Argmin-Sum Pattern

L	n	r	Mean	Variance	$>95\%$	$>99\%$
4	8	0.25	0.11	0.06	0.67	1.05
		0.50	0.25	0.17	1.15	1.67
		0.75	0.42	0.33	1.62	2.24
4	12	0.25	0.14	0.09	0.83	1.30
		0.50	0.30	0.25	1.42	2.07
		0.75	0.51	0.49	1.98	2.75
4	16	0.25	0.16	0.12	0.96	1.51
		0.50	0.35	0.34	1.64	2.40
		0.75	0.59	0.66	2.29	3.18
6	8	0.25	0.14	0.07	0.74	1.10
		0.50	0.30	0.20	1.25	1.75
		0.75	0.52	0.38	1.75	2.34
6	12	0.25	0.17	0.10	0.91	1.36
		0.50	0.37	0.30	1.54	2.16
		0.75	0.63	0.57	2.15	2.89
6	16	0.25	0.20	0.14	1.06	1.59
		0.50	0.43	0.40	1.78	2.51
		0.75	0.73	0.76	2.49	3.35
8	8	0.25	0.16	0.07	0.78	1.12
		0.50	0.34	0.21	1.31	1.79
		0.75	0.58	0.40	1.83	2.41
8	12	0.25	0.19	0.11	0.96	1.40
		0.50	0.42	0.32	1.61	2.22
		0.75	0.71	0.61	2.25	2.97
8	16	0.25	0.22	0.15	1.11	1.63
		0.50	0.48	0.43	1.86	2.57
		0.75	0.82	0.82	2.60	3.44

Table 1. Argmin-Sum Simulation Numbers

3.3.1 Simulation

To begin, we use a simulation to analyze the effect of perforating the inner loop (the `j` loop) in the argmin-sum pattern. The simulation models the `f(i,j)` as random variables $X_{i,j}$ chosen as i.i.d samples from a uniform distribution on the interval $[0, 1]$. The argmin-sum computation searches a sequence of sums for the minimum sum. The original computation produces the index of the sequence with the minimum sum as its result. The perforated computation produces the index of the sequence with the minimum perforated sum. The perforation noise $|D|$ is the difference between the minimum sum from the original computation and the complete sum (not the perforated sum) of the sequence whose index the perforated computation produces.

Table 1 presents the results from this simulation. The perforation rate is $\frac{1}{2}$. Each row in the table contains the results of a single simulation run. The first three columns in each row present the values of `L`, `n`, and perforation rate r for the simulation. Column 4 (Mean) presents the mean of the absolute perforation noise. Column 5 (Variance) presents the variance of the absolute perforation noise. Column 6 ($>$95%) presents the 95th percentile, i.e., the value below which 95% of the recorded perforation noise values fall. Column 7 ($>$ 99%) presents the 99th percentile. These reported percentiles are estimators for the true percentiles, and can thus be used to bound the occurrence of large perforation noise.

We can use the bounds from Columns 6 and 7 to calculate bounds for the case when the inputs X_i are i.i.d. samples from a uniform distribution on an arbitrary interval $[a, b]$. E.g., we can estimate the extrapolated bound β' from the corresponding bound β by $\beta' = (b - a)\beta + a$.

3.3.2 Assumptions

For each $i \in \{1, \ldots, L\}$, we assume that $X_{i,j} = \mathtt{f(i, j)}$ are conditionally independent and drawn from a distribution F_i. Moreover, we assume that the distributions F_i themselves are independent and distributed according to some distribution G. The perforation vector P coordinates take only the values from the set $\{0, 1\}$.

3.3.3 Analysis

In the argmin-sum pattern, we compute an index which is then used later in the program. We model the value of an index i as the *entire* sum $X_i = \sum_{j=1}^{n} X_{i,j}$.

Therefore, the original computation produces the value

$$S_O = \min_i A' X_i = \min_i \sum_{j=1}^{n} X_{i,j}, \tag{62}$$

while the perforated computation produces the value

$$S_P = \sum_{j=1}^{n} X_{\gamma,j}, \tag{63}$$

where

$$\gamma \equiv \arg\min_i \sum_{j=1}^{m} X_{i,j} \tag{64}$$

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 any loss of generality, choose a so-called truncation perforation vector.

We are interested in studying the perforation noise

$$D \equiv S_P - S_O \geq 0 \tag{65}$$

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

$$\begin{aligned} D &= Y_\gamma + Z_\gamma - Y_\omega - Z_\omega && (66)\\ &\leq Y_\gamma + Z_\gamma - \min_i Y_i - \min_i Z_i && (67)\\ &= Z_\gamma - \min_i Z_i. && (68) \end{aligned}$$

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 the perforation noise for this pattern is non-negative because $Z_\gamma \geq \min_i Z_i$.

To obtain the mean and the variance of the perforation noise we first consider the case where all the distributions F_i are equivalent (i.e., $F_i = F$ for some distribution F). We can express this by assuming that the distribution (on distributions) G is a degenerate distribution that assigns probability one to a single value, namely F. We write this as $G = \Delta_F$. This situation is an extreme example of the situation where the F_i are all very similar and have overlapping support. In such a situation, it is hard to distinguish between samples from F_i and F_j. Therefore, we will call this the *overlapped* case.

Define Z to the sum of $n-m$ independent F-distributed random variables. Then Z_i has the same distribution as Z, and, moreover, γ is independent of Z_γ and, in particular, Z_γ has the same distribution as Z. Therefore,

$$\mathbb{E}(\bar{D}) = \mathbb{E}(Z) - \mathbb{E}(\min_i Z_i), \tag{69}$$

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).

We will now consider a different case: Assume that

$$G = \alpha\, \Delta_F + (1-\alpha)\, G', \tag{70}$$

where $0 < \alpha < 1$, where F is a distribution on the real line and where G' is some distribution (on distributions) such that:

$$\exists \theta \in \mathbb{R},\ F(-\infty, \theta) = 1 \text{ and } G'(\{F' : F'(\theta, \infty) = 1\}) = 1. \tag{71}$$

That is, random values sampled from the distribution F are always smaller than those sampled from distributions F' (that are sampled from G'). Again, let Z be the distribution of the sum of $n - m$ independent F-distributed random variables. We will further assume that

$$1 - (1-\alpha)^L > 1 - \epsilon \tag{72}$$

for small $\epsilon \approx 0$. That is, with high probability, one of the rows i will be sampled from F (i.e., $F_i = F$ for some i with high probability.) We will call this the *separated* case. Under these assumptions, Eq. (69) holds approximately (up to a term bounded by ϵ.), but now the second term will be the first order statistic *of an expected size* $\alpha L < L$ *sample*. If F has a light left tail and αL is small, then the order statistic will not be very different from the mean, and we can therefore expect the bound on the perforation noise to be much reduced compared to the overlapped case.

To understand why the perforated computation excels in the separated case, note that we can think of the perforated sum on each step i as a quick test of the location of the mass of the distribution F_i. If there is a distribution that is separated from the rest and concentrates its mass on smaller values, the smallest perforated sum will identify this distribution whenever it is present and the perforated computation will correctly choose to use the corresponding index in those cases. In the separated case, the distribution F has precisely this property and the condition (72) ensures that at least one row is sampled from F. In particular, the smallest perforated sum will with probability $1 - \epsilon$ identify a stage i where $F_i = F$.

The former analysis also suggests situations where we can expect this perforated computation to not produce satisfactory results. If the P_i are overlapped and have very long tails, then the perforated sums will not be good approximations to the actual summations, as it is likely that a very large summand will be missed in the perforated sum.

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

Define $M_m = \min_{i \le m} Z_i$. Then $\frac{1}{w}(M_m - a + \frac{w}{2})$ has a Beta$(1, m)$ distribution, and so M_m has mean

$$a + \frac{w}{m+1} - \frac{w}{2} \tag{73}$$

and variance

$$\frac{m w^2}{(m+1)^2(m+2)}. \tag{74}$$

From (69), we have

$$\mathbb{E}(\bar{D}) = \mathbb{E}(Z) - \mathbb{E}(M_L) \tag{75}$$

$$= \frac{w}{2} - \frac{w}{L+1}. \tag{76}$$

[2] Z will never 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. In fact, by a central limit theorem argument, we would expect the sum Z to converge in distribution to a Gaussian. In this case, we might take our approximation to cover some number of standard deviations.

Furthermore, as γ is independent of every Z_i, it follows that Z_γ is independent of M_L. Therefore,

$$\mathrm{var}(\bar{D}) = \mathrm{var}(Z) - \mathrm{var}(M_L) \tag{77}$$

$$= \frac{1}{12} w^2 + \frac{L w^2}{(L+1)^2(L+2)}. \tag{78}$$

Limit cases agree with our intuition: as $L \to \infty$, we have that $\mathbb{E}(\bar{D}) \to \frac{w}{2}$ and $\mathrm{var}(\bar{D}) \to \frac{1}{12} w^2$. 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 one-sided Chebyshev's inequality (Eq. 19), it follows that with probability at least $1 - \epsilon$

$$|D| < \sqrt{\mathrm{var}(\bar{D})\Big(\frac{1}{\epsilon} - 1\Big)} + \mathbb{E}\bar{D} \tag{79}$$

An analysis of bounds in the separated case is left to future work.

3.4 Sum-Division Pattern

We present an example of the original and the transformed code for the Sum-Division computation pattern in Figure 8 .

Original code	Transformed Code

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

Figure 8. Sum-Division Pattern

3.4.1 Assumptions

Let $X_i = \mathtt{x}(i)$ and $Y_i = \mathtt{y}(i)$ denote random variables representing the values of the inner computations. We will 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 $\mathrm{var}(Z|Y) = \frac{\sigma_Z^2}{Y}$.

The model computation in our minds that justifies these assumptions is one where on each iteration some number of tests Y are performed, and that each of these tests is a noisy measurement, and so their average X/Y is a better estimate. The outer loop might gauge the accuracy in order to run more tests, but we do not model that aspect here.

3.4.2 Analysis

The perforation vector coordinates only take the values from the set $\{0, 1\}$. Note that the independence of pairs of random variable implies that the perforation strategy does not influence the final result. To simplify the derivation, but without loss of generality, we represent the perforation noise as the sum of first m elements.

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

$$S_O = \frac{X_1^n}{Y_1^n} = \sum_{i=1}^n \frac{Y_i}{Y_1^n} Z_i, \tag{80}$$

while the value of the perforated computation is given by

$$S_P = \sum_{i=1}^m \frac{Y_i}{Y_1^m} Z_i, \tag{81}$$

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 will begin by studying the (signed) perforation noise

$$D \equiv S_P - S_O. \tag{82}$$

We have that the conditional expectation of D given $Y_{1:n} = \{Y_1, \ldots, Y_n\}$ satisfies

$$\mathbb{E}(D|Y_{1:n}) = \sum_{i=1}^{n} \frac{Y_i}{Y_1^n}\mu - \sum_{i=1}^{m} \frac{Y_i}{Y_1^m}\mu \tag{83}$$

$$= \mu\left(\sum_{i=1}^{n} \frac{Y_i}{Y_1^n} - \sum_{i=1}^{m} \frac{Y_i}{Y_1^m}\right) \tag{84}$$

$$= 0 \tag{85}$$

and that the conditional variance satisfies

$$\mathrm{var}(D|Y_{1:n}) = \mathrm{var}\left(\sum_{i=1}^{m} Z_i Y_i \left(\frac{1}{Y_1^m} - \frac{1}{Y_1^n}\right)\right) \tag{86}$$

$$+ \mathrm{var}\left(\sum_{i=m+1}^{n} Z_i \frac{Y_i}{Y_1^n}\right) \tag{87}$$

$$= \sigma_Z^2\left(\sum_{i=1}^{m} Y_i \left(\frac{1}{Y_1^m} - \frac{1}{Y_1^n}\right)^2 + \sum_{i=m+1}^{n} \frac{Y_i}{Y_1^n}\frac{1}{Y_1^n}\right) \tag{88}$$

$$= \sigma_Z^2\left(\frac{1}{Y_1^m} - \frac{1}{Y_1^n}\right) \tag{89}$$

By the law of iterated expectations, we have

$$\mathbb{E}(D) = \mathbb{E}(\mathbb{E}(D|Y_{1:n})) = 0. \tag{90}$$

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

$$\mathrm{var}(D) = \mathbb{E}(\mathrm{var}(D|Y)) \tag{91}$$

$$= \frac{\sigma_Z^2}{\theta}\left(\frac{1}{m\alpha - 1} - \frac{1}{n\alpha - 1}\right). \tag{92}$$

Again, using Chebyshev's inequality, we can bound the probability of large absolute perforation noise $|D|$ using (17).

3.5 Worst Case Analysis of the Patterns

In this section we derive the expressions for the worst case perforation noise of the result of the perforated computation from the result of the original, unperforated application. We assume that each summation term x_i belongs to a finite interval, i.e. $x_i \in [a, b]$. We are looking for the maximum value of the absolute perforation noise

$$D_{WC} = \max_{x_i} |D(x_1, x_2, \ldots x_n)| \tag{93}$$

Note that, while we use the expressions for D from the Section 3, we have replaced all random variables X_i with their deterministic counterparts x_i, making the expressions D deterministic. In some of the following analyses we make additional assumptions on the perforation strategy to simplify the algebraic expressions.

3.5.1 Sum of Independent Variables

The perforation noise between the original and the perforated computations is defined by (27) (Section 3.1).

$$D = \sum_{i:P_i \neq 0} (P_i - 1)\, x_i - \sum_{j:P_j = 0} x_j \tag{94}$$

If all non-zero coordinates of the vector P have the same value $\frac{n}{m}$, the perforation noise is equal to

$$D = \Big(\frac{n}{m} - 1\Big) \sum_{i:P_i \neq 0} x_i - \sum_{j:P_j = 0} x_j \tag{95}$$

$$= S_+ - S_- \tag{96}$$

Since the function is monotonously increasing, the maximum of $|D|$ can be obtained by maximizing the distance between sums S_+ and S_-. We assume the finite range for the values of every input $x_i \in [a, b]$, where $a, b \in \mathbb{R}$. Maximum distance between S_+ and S_- is obtained if all terms in one sum are equal to a, and all terms in the other sum are equal to b. In this case the absolute perforation noise becomes

$$D_{WC} = \Big|\Big(\frac{n}{m} - 1\Big) mb - (n - m)\, a\Big| \tag{97}$$

$$= \Big|\Big(\frac{n}{m} - 1\Big) ma - (n - m)\, b\Big| \tag{98}$$

$$= (n - m)\,(b - a) \tag{99}$$

If the perforation strategy is interleaving with $k = 2$, then $m = \frac{n}{2}$, and the perforation noise is equal to $\frac{n}{2}(b - a)$.

3.5.2 Sum of Random Walk Components

We defined the perforation noise between the original and the perforated computations in (27) (Section 3.1).

$$D = \sum_{i:P_i \neq 0} (P_i - 1)\, x_i - \sum_{j:P_j = 0} x_j \tag{100}$$

We can express the input variable x_i as $x_i = x_0 + \sum_{i=1}^{n} \delta_i$. We assume that the variable $x_0 = x$ and $\delta \in [-d, d]$ $(a, b \in \mathbb{R})$. By replacing the values of x_i in (100) and using the previous identity, the perforation noise becomes

$$D = \sum_{i:P_i \neq 0} (P_i - 1) \sum_{j=1}^{i-1} \delta_j - \sum_{i:P_i = 0} \sum_{j=1}^{i-1} \delta_j \tag{101}$$

Note that in the previous equation, the terms including x sum up to 0, which follows from the definition of the vector P:

$$\sum_{i:P_i \neq 0} P_i x - \sum_{i=1}^{n} x = x\Big(\sum_{i:P_i \neq 0} P_i - n\Big) = 0 \tag{102}$$

Note that it is possible for a particular term δ_i to appear in both the left and the right summation, which makes the optimization dependent on the perforation strategy to specify the coefficients of each δ_i. The expression for variance will also depend on the selection of perforation strategy.

As an illustration, we will derive the expression for the maximum perforation noise for the interleaving perforation strategy with rate $k = 2$, which implies $P_{2i-1} = 2$ and $P_{2i} = 0$. Let set $Q_n = \{2t+1 : 0 < 2t+1 \leq n \wedge t \in N_0\}$ denote the odd-indexed iterations, and set $R_n = \{2t : 0 < 2t \leq n \wedge t \in N_0\}$ denote the even-indexed iterations of the loop. The perforation noise then takes the form

$$D = \sum_{i \in Q_n} \sum_{j=1}^{i-1} \delta_j - \sum_{i \in R_n} \sum_{j=1}^{i-1} \delta_j \tag{103}$$

$$= \sum_{i \in Q_n} \Big(\sum_{j=1}^{i-1} \delta_j - \sum_{j=1}^{i} \delta_j\Big) \tag{104}$$

$$= -\sum_{i \in Q_n} \delta_i \tag{105}$$

Application	Computation	Location	Exec. Time %	Mean Iterations	Executions
bodytrack	`ImageErrorEdge`	`ImageMeasurements.cpp`, 122	26.7%	40	400391
swaptions	`HJM_Swaption_Blocking`	`HJM_Swaption_Blocking.cpp`, 157	100%	1250	64
streamcluster	`pFL`	`streamcluster.cpp`, 600	98.5%	52	49
x264	`refine_subpel`	`me.c`, 665	16.3%	8.30	1339415

Table 2. Execution Statistics for Computations

The range of values of $|\delta|$ is $[0, d]$. The absolute perforation noise reaches its maximum when all δ_i have the value d

$$D_{WC} = \frac{n}{2}d \tag{106}$$

Note that (105) implies that the only relevant differences are those at odd positions $\delta_{2i-1} = x_{2i} - x_{2i-1} = d$. The differences $x_{2i+1} - x_{2i}$ may have an arbitrary value from the assumed range $[-d, d]$. Examples of potential worst case scenarios include adding d to the previous term in every loop iteration, subtracting d from the previous term in every loop iteration, or interchangeably adding and subtracting d in adjacent iterations.

3.5.3 Mean Pattern

To derive the worst case analysis result for mean pattern, we can utilize the results from the worst case sum analysis. Since $M_n = \frac{1}{n}S_n$, the perforation noise of the mean of n elements is proportionally smaller than the perforation noise of the sum. For example, for the case when the summands are independent it can be derived that

$$D_{WC} = \left(1 - \frac{m}{n}\right)(b - a) \tag{107}$$

3.5.4 Argmin-Sum Pattern

Recall the perforation noise as given by Equations 65 and 66. We assume that each term x_i of the sums S_P and S_O take value from a range $[a, b]$.

To maximize the perforation noise D we can maximize the differences $D_Y = Y_\gamma - Y_\omega$ and $D_Z = Z_\gamma - Z_\omega$ independently. Since the sum Y_γ is defined to be the minimum sum, the difference D_Y reaches a maximum value 0 if $Y_\gamma = Y_\omega$. The difference D_Z is maximized when all terms in the sum Z_γ take the value b, and all terms in the sum Z_ω take the value a. Since these sums have $n - m$ elements, the maximum perforation noise is

$$D_{WC} = (n - m)(b - a). \tag{108}$$

3.5.5 Sum-Division Pattern

To perform the worst case analysis for Sum-Division pattern, we bound the potential values of the variables $x_i \in [a, b]$ and $y_i \in [c, d]$. The perforation noise between the the results of the perforated and the original computations is defined by Equation 82 (Section 3.4).

If the interval $[c, d]$ contains value 0, then the worst case perforation noise approaches infinity. We can choose the terms y_i in a way that the first m terms sum up to 0 (i.e., $\sum_{i=0}^{m} y_i - 0$, but, remaining summands can be chosen such that the sum $\sum_{i=m+1}^{n} y_i$ does not equal to 0. As a consequence, the perforated division computation will have an infinite value, unlike the original division computation.

If the interval $[c, d]$ does not contain value 0, then finding the maximum perforation noise becomes a non-trivial non-linear optimization problem. In general, we must resort to numerical methods to find the worst-case perforation noise. As an illustration, if $x_i \in [15.0, 25.0]$ and $y_i \in [1000, 1200]$, then the maximum perforation noise for $n = 10$ equals 0.038.

4. Experimental Results

We evaluate the probabilistic analyses from Section 3 on applications from the PARSEC benchmark suite [4]. In this section we outline the methodology we used for our evaluation.

We use four computations which Quality of Service Profiling [11] identified as good optimization candidates. These computations consume a significant amount of the execution time and can trade off accuracy for additional performance gains. For each subcomputation we identify a set of applicable analyses. Each computation comes from a different benchmark application:

1. The `ImageErrorEdge` computation comes from bodytrack, a machine vision application. We identified that this subcomputation is an instance of both the sum and the sum division patterns.
2. The `HJM_Swaption_Blocking` computation comes from swaptions, a financial analysis application. We identified that this subcomputation is an instance of the mean pattern.
3. The `pFL` computation comes from streamcluster, an unsupervised learning application. We identified that this subcomputation is an instance of the sum pattern.
4. The `refine_subpel` computation comes from x264, a video encoder. We identified that this subcomputation is an instance of the argmin-sum pattern.

The organization of this section is as follows. We first discuss the identification of the patterns in Section 4.1. We describe the experimental setup in Section 4.2. In Section 4.3 we summarize the experimental results. In Sections 4.4-4.7 we discuss results from individual subcomputations in greater detail. In Section 4.8 we compare the results from the probabilistic analyses against the corresponding worst-case analyses.

4.1 Computation Identification

We use Quality of Service Profiling [11] to identify the loops to perforate. The Quality of Service Profiler perforates different loops in turn. For each perforation it runs the application on representative inputs, recording the execution time and output. It identifies loops whose perforation causes a combination of improved performance and small output changes as potentially profitable optimization targets.

The Quality of Service Profiler also produces execution statistics for the perforated loops. These statistics include the number of times the loop was executed, the average number of iterations per loop execution, and the minimum and maximum number of loop iterations in a single execution. We use these loop execution statistics to calculate the parameters of the probabilistic analyses.

Table 2 presents the execution statistics for the loops that we perforate. The first two columns present the application and computation names. The third column (Location) presents the file name and the line where the loop begins. The fourth column (Execution Time) presents the percentage of LLVM bitcode instructions performed within the computation. The fifth column (Average Iterations) presents the mean number of iterations that the loop executes. This number usually coincides with the number of inputs consumed

Application	Pattern	Input Distribution	Output	Perforation Rate	Output Noise Predicted	Output Noise Observed
bodytrack	sum	Normal (μ = 6.25, σ = 4.32)	249.97 ($\pm$ 22.37)	0.25	12.59 ($\pm$ 9.51)	8.400 ($\pm$ 6.050)
				0.5	22.13 ($\pm$ 16.47)	22.03 ($\pm$ 14.31)
				0.75	38.34 ($\pm$ 28.96)	79.15 ($\pm$ 30.20)
bodytrack	sum division	Gamma (α = 12.08, θ = 2.13, σ_Z^2 = 0.57)	0.240 ($\pm$ 0.022)	0.25	0.000 ($\pm$ 0.014)	0.001 ($\pm$ 0.009)
				0.50	0.000 ($\pm$ 0.024)	0.007 ($\pm$ 0.016)
				0.75	0.000 ($\pm$ 0.041)	0.032 ($\pm$ 0.025)
swaptions	mean	Normal (μ = 0.05, σ = 0.05)	0.048 ($\pm$ 0.047)	0.25	0.0007 ($\pm$ 0.001)	0.0001 ($\pm 5 \cdot 10^{-5}$)
				0.50	0.0011 ($\pm$ 0.001)	0.0001 ($\pm 5 \cdot 10^{-5}$)
				0.75	0.0020 ($\pm$ 0.001)	0.0002 ($\pm 5 \cdot 10^{-5}$)
streamcluster	sum	Normal (μ = 87.81, σ = 1237.91	6059 ($\pm$ 10510)	0.25	4874 ($\pm$ 3683)	5244 ($\pm$ 7305)
				0.50	8324 ($\pm$ 6289)	9386 ($\pm$ 26331)
				0.75	14349 ($\pm$ 10841)	17934 ($\pm$ 30083)
x264	argmin sum	Uniform (a = 421.5, w = 843.0)	654.7 ($\pm$ 662.0)	0.25	301.1 ($\pm$ 264.8)	12.53 ($\pm$ 23.33)
		Uniform (a = 605.9, w = 1211.8)		0.50	432.9 ($\pm$ 380.7)	14.70 ($\pm$ 26.15)
		Uniform (a = 892.8, w = 1785.6)		0.75	637.9 ($\pm$ 560.9)	25.33 ($\pm$ 41.57)

Table 3. Comparison of the Observed and the Predicted Perforation Noise Means and Variances.

by a single execution of the computation. Finally, the sixth column (Executions) presents the number of times the loop was executed. This number usually corresponds to the number of the outputs that the computation produces during the lifetime of the application. We use these figures to calculate parameters to plug into the expressions for the perforation noise from the probabilistic analysis and the worst-case analysis.

4.2 Experimental Setup

For each subcomputation we record the values of the inputs and outputs for the subcomputation during the execution of the program on representative inputs. We perforate each subcomputation using the sampling perforation strategy with three perforation rates — 0.25, 0.50, and 0.75.

To run the applications, we used the representative inputs that came with the benchmark suite, or augmented the suite with additional inputs from reference data repositories. For bodytrack, we used part of the `sequenceA` input provided by benchmark developers. For swaptions, we used the `simlarge` input that comes with the benchmark suite. For streamcluster, we used 10^5 points from the `animalNorm` dataset from UCI Machine Learning Repository [2] For x264, we used a sequence of 30 frames from the `tractor` video sequence from Xiph.org Foundation web site.

We perform the following steps for the analysis of each loop:

- **Pattern Identification:** We identify the pattern from Section 3 that each loop implements. The probabilistic analysis of this pattern gives us parameterized expressions for the *predicted mean* and *predicted variance* of the perforation noise.
- **Maximum Likelihood Parameter Fit:** We use the recorded execution statistics to obtain numerical values for the symbolic parameters of the predicted mean and predicted variance of the perforation noise. Starting with the assumed probability distributions, we use the maximum likelihood fit for the distributions to the recorded execution statistics.
- **Predicted Mean and Variance:** Using the maximum likelihood fit, we compute the predicted mean and variance of the perforation noise for each of the perforated loops.
- **Observed Mean and Variance:** Using the recorded output statistics, we compute the *observed mean* and *observed variance* of the perforation noise for each of the perforated loops.
- **Comparison:** We compare the predicted mean and variance with the observed mean and variance to determine how well the predicted mean and variance predict the observed mean and variance. We discuss potential reasons for mismatches between the predicted and observed values.
- **Bounds:** Under the assumption that the inputs are selected from a finite range, we calculate a bound on the perforation noise that is, at least 99% of the time, larger than the actual observed perforation noise (i.e., $\epsilon = 0.01$). A comparison of this bound with the bound from the worst-case analysis provides insight into how accurately the two methods predict the perforation noise.

We perform the experiments on Intel Xeon E5520 Dual Quadcore processor, running Ubuntu 10.10. We use LLVM 2.7 to compile the benchmark applications. Our prototype analyzer is implemented in Python, with parts implemented in C++ and R.

4.3 Overall Results

We next present experimental results that characterize the accuracy of our moment-based and bounds-based analyses of the perforation noise.

4.3.1 Moment-Based Analysis

Table 3 presents the expected value and the variance of a) the subcomputation input, b) the subcomputation output, c) the perforation noise (all are recorded from the application execution) and d) the perforation noise predicted by the probabilistic analysis. Column 1 of Table 3 (Application) presents the name of the benchmark application. Column 2 (Pattern) presents the identified pattern from Section 3. Column 3 (Input Distribution) presents the parameters of the assumed input distribution as produced by the maximum likelihood fit. Column 4 (Output) presents the mean (the first number) and the standard deviation (the second number, in parentheses) of the output of the original (unperforated) computation, as calculated from the recorded output values from the running application. Column 5 (Perforation Rate) presents the perforation rate of the loop for which the results were obtained. Column 6 (Predicted Perforation Noise) presents the mean (the first number) and the standard deviation (the second number) of the predicted perforation noise D. We use the expressions from Section 3 to calculate this mean and standard deviation. Finally, Column 7 (Observed Perforation

Application	Pattern	Input Range	Perforation Rate	Observed Noise $> 95\%$	Observed Noise $> 99\%$	Observed Noise Maximum	Predicted Noise $> 99\%$
bodytrack	sum	$[a, b] = [0.0, 20.0]$	0.25	19.74	25.05	47.99	145.6
			0.50	47.98	59.02	95.90	205.9
			0.75	130.2	148.9	218.7	252.2
bodytrack	sum division	$[a, b] = [0, 20]$ $[c, d] = [1000, 1200]$	0.25	0.018	0.0231	0.0424	0.1362
			0.50	0.034	0.0437	0.0763	0.2361
			0.75	0.074	0.0897	0.1422	0.4097
swaptions	mean	$[a, b] = [0.0, 0.2]$	0.25	0.00023	0.00024	0.00024	0.0053
			0.50	0.00015	0.00015	0.00015	0.0092
			0.75	0.00022	0.00026	0.00026	0.0159
streamcluster	sum	$[a, b] = [0, 12371]$	0.25	1144	18805	18805	102670
			0.50	9542	18805	91252	145198
			0.75	18805	19623	28698	177830
x264	argmin-sum	$[a, b] = [0, 490.1]$	0.25	50.00	105.0	1046	2935
			0.50	58.00	119.0	1046	4220
			0.75	97.00	197.0	1501	6218

Table 4. Comparison of the Observed and the Predicted Perforation Noise Bounds.

Noise) presents the mean and standard deviation[3] of the observed perforation noise (as calculated using the observations from the perforated and original computations).

For four out of five benchmarks (except streamcluster) a comparison between the mean output value (fourth column) and the mean observed perforation noise (seventh column) indicates that in most cases the mean perforation noise is less than 10% of the mean output value. The perforation noise is greater than 10% only for x264 at the perforation rate 0.75 and bodytrack sum at the perforation rate 0.75.

Three of the five theoretical analyses (bodytrack sum division, swaptions, and x264) conservatively predict the mean and variance of the perforation noise. The fourth analysis, bodytrack sum analysis, provides a conservative prediction for perforation rates 0.25 and 0.5. For these two cases the predicted perforation noise is up to 50% greater than the corresponding mean observed perforation noise. For swaptions and x264, the predicted perforation noise is more than a factor of 10 greater than the mean observed perforation noise.

We attribute the discrepancy between the predicted and mean observed perforation noise to the analysis assumptions about the inputs of the computations. The assumptions include independence properties between input components and distribution assumptions of the inputs such as normality or uniformity. These assumptions, as we will see in the detailed application results, may or may not hold for the data sets that were recorded during program executions. We nevertheless apply the analysis to the computation even if some of the initial assumptions may not be satisfied, given that many probabilistic analyses are robust with respect to moderate departures from the distribution-related assumptions. For two analyses (bodytrack sum for perforation rate 0.75 and streamcluster), we discuss potential reasons why the predicted perforation noise is smaller than the observed perforation noise (see Sections 4.4 and 4.6).

For each benchmark we present a histogram of the input data, together with the fitted analytical distribution (Figures 10, 11, 13, 15, 17). For each figure, the X-axis corresponds to the binned values of the input. The Y-axis presents the frequency density of the inputs, with each box representing the frequency of a range of input values. The area covered by the histogram is equal to 1. The red curve represents the density of the fitted distribution. We present the parameters of the fitted distributions in the third column of Table 3.

Often, the empirical distributions are not symmetric. We use the *skewness* to characterize the departure of the distribution from assumed symmetry. The skewness coefficient represents the third moment of the distribution, $g = \mathbb{E}\left[\left(\frac{X-\mu}{\sigma}\right)^3\right]$. The absolute value of g indicates the amount of asymmetry (with 0 indicating a symmetrical distribution). The sign of g indicates the direction of asymmetry. Values of g greater than 0 indicate that the distribution is right-skewed, i.e., its right tail is heavier than its left tail and most of the probability mass is located left from the mean value. Similarly, values of g smaller than 0 indicate a left-skewed distribution.

4.3.2 Bound-Based Analysis

Table 4 presents the observed and the predicted bounds on the perforation noise. The first and the second columns present the name of the application and the analyzed pattern. Column 3 (Input Range) bounds the interval of input variables for the calculation of the bound according to Hoeffding's inequality, where applicable. In particular, for every benchmark we choose the interval to cover over 95% of the probability mass. Column 4 presents the perforation rate. The following three columns present the perforation noises that we observed from the application executions. Column 5 presents the perforation noises that are greater than 95% of the observed noises, Column 6 presents the perforation noises that are greater than 99% of the observed noises, and Column 7 presents the maximum observed noise. Finally, Column 8 presents the predicted perforation noise bound that is greater than at least 99% of the perforation noises.

We calculate the predicted perforation noise bound at level 99% using the equations derived from Hoeffding's and Chebyshev's inequalities presented in Section 3. For three patterns (bodytrack sum, swaptions mean and streamcluster sum) we calculate the bound from the Hoeffding's inequality. To calculate the upper bound, we use the bounds on the input variables specified in Column 3, while setting $\epsilon = 0.01$. For the bodytrack sum-division pattern, we calculate a two-sided Chebyshev bound, and in addition use the input distribution parameters (α, θ, σ_Z^2) reported in Table 3. For the x264 argmin-sum pattern, we calculate a one-sided Chebyshev bound, using the input distribution parameter w reported in Table 3.

[3] This is the standard deviation of the distribution. The standard error of the noise mean can be calculated as $\sigma/\sqrt{v}$, where v is the number of loop executions

For all benchmarks, the noise predicted by the theoretical bound at level 99% was greater than the observed perforation noise at the same level. Moreover, the predicted noise was greater than the maximum observed noise in all cases. That is, the predicted noise is greater than 100% of the observed perforation noises. We attribute the conservative bound results to the assumptions that we made about the properties of the inputs, including the input intervals, the independence of the inputs. In the cases where we use Chebyshev's bound (bodytrack sum-division and x264 argmin-sum) the distribution assumptions on the inputs, used to calculate the variance also contribute to the conservative estimates.

4.4 Bodytrack

Bodytrack is a machine vision application which monitors the motion of human body captured by multiple security cameras. Bodytrack uses annealed particle filtering to locate body parts and analyze their motion across multiple frames.

The identified loop is the inner loop in the `ImageErrorEdge` function. This loop is located in the `ImageMeasurements.cpp` file, starting at line 122. The loop is a part of a larger computation that approximates a multimodal probability distribution using Markov Chain Monte Carlo simulation. The loop calculates the values of the sampled distribution function that are contributed by *edges*, which are borders between image regions with distinct brightness levels.

```
float error = 0;
int samples = 0;

for(int i = 0; i < ImageMaps.size(); i++){
  // Perforatable loop:
  for(int j = 0; j <  ProjBodies[i].Size(); j++)
      error += addError(Body[i][j], ImageMap[i], Positions[j]);
      samples += addSample(Body[i][j], ImageMap[i], Positions[j]);
  }
}

float div = error / samples;
```

Figure 9. Bodytrack Computation

Figure 9 presents a simplified version of the computation. We perforate the loop with induction variable `j`. In each iteration the values of the variables `error` and `samples` are updated with new contributions from the functions `addError` and `addSample`.

Sum Pattern Analysis. The variables `error` and `samples` are incremented within the loop. The computation produces both values as its output. We can apply the sum pattern analysis (Section 3.1) independently on both variables. Since the sum pattern analysis for both variables yields similar results, we will concentrate in this section on the analysis for the `error` variable.

We present the results of the probabilistic analysis of the perforated computation for the variable `error` in Table 3, Row 1. Because of the extrapolation that occurs after the loop executes, the mean observed perforation noise is smaller than 10% of the mean output of this computation. In addition, the mean observed perforation noise is, on average, smaller than the predicted perforation noise.

The analysis assumes that the inputs are normally distributed. Figure 10 presents the histogram of the densities of the actual inputs, together with the fitted distribution. The histogram suggests that the empirical distribution is non-symmetrical, with support bounded from below at 0. The empirical distribution of `errors` has skewness coefficient $g = 1.24 > 0$, indicating right-skewness and longer right tail of the distribution.

A comparison of the predicted perforation noise assuming normally distributed inputs and the mean observed perforation noise

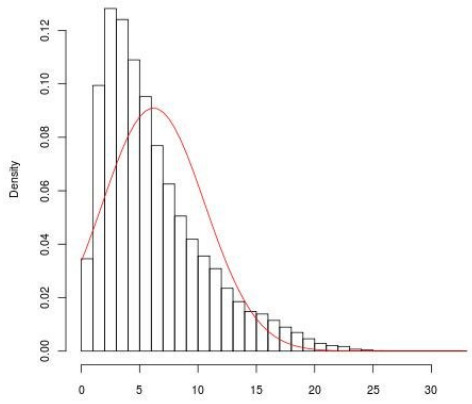

Figure 10. Histogram of values returned by `addError` and fitted normal distribution from Sum analysis

shows that the predicted noise tends to conservatively overestimate the observed noise for the perforation rates 0.25 and 0.50. For the perforation rate 0.75, the observed noise is greater than the predicted noise. We attribute the imprecision of the prediction to the sensitivity of the analysis to the distribution of the sum terms of the variable `errors`. The true distribution of the input terms is not normal (it has only positive values), and the observed mean noise increases at a higher rate with the perforation rate increases than the predicted mean noise.

Sum Division Pattern Analysis. The `ImageErrorEdge` computation can also be analyzed as a *sum division* pattern. As we can see in Figure 9, the computation following the loop divides the sum of model errors by the sum of samples. We can use the *sum division* pattern (Section 3.4) to analyze this computation. The values calculated in the loop (`error` and `samples`) serve as inputs to the division operator.

Table 3, Row 2 presents the results of the probabilistic analysis of the perforated computation for the variable `div`. The observed accuracy loss of the division computation (this is the observed perforation noise divided by the mean output) is somewhat smaller than the observed accuracy losses of the individual numerator and denominator, especially when more iterations are dropped — for the perforation rate 0.75 the accuracy loss of variable `div` analyzed in sum division pattern is 6.7% of the original value, compared to 10% for the observed accuracy loss of the variable `error` analyzed using the sum pattern.

The analysis makes a few assumptions about the distribution of the inputs (Section 3.1). We obtain the value of the parameter σ_Z^2 from the recorded values of the variables Z_i and Y_i. Given a random variable $W_i = Z_i\sqrt{Y_i}$, $\sigma_Z^2 = \mathrm{var}(W_i)$. This value is consistent with the assumption that the $\mathbb{E}(Z|Y) = \frac{\sigma_Z^2}{Y}$. To obtain the value of W_i our instrumentation records the values of Y_i and Z_i in the log file. The analyzer uses these values to calculate W_i.

The output of the function `addError` corresponds to X_i and the output of the function `addSample` corresponds to Y_i. A comparison of the predicted perforation noise assuming the gamma distribution on Y_i shows that the predicted noise tends to conservatively overestimate the observed noise.

Figure 11 presents the distribution of the random variable Y. Although the fitted input distribution does not completely match the observed distribution, the results of the analysis are conser-

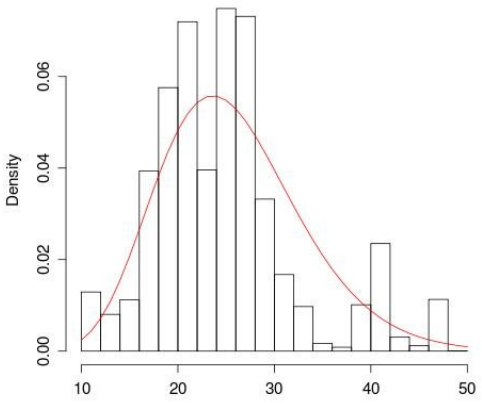

Figure 11. Histogram of variable `samples` and fitted gamma distribution from Sum division analysis

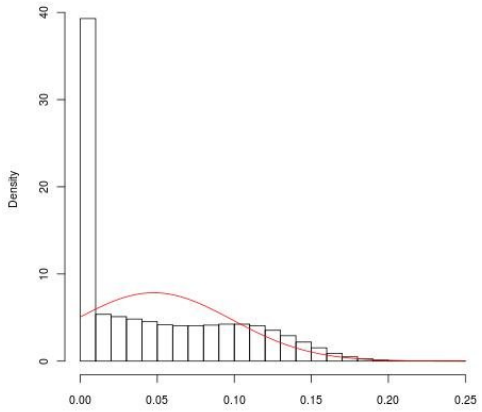

Figure 13. Histogram of variable `simres` and fitted normal distribution from Mean analysis

vative. The predicted mean of the noise is equal to 0. The means of the observed noise deviate by less than one standard deviation from the predicted mean. Furthermore, the standard deviation of the observed noise is smaller than the standard deviation of the predicted noise. And the interval of three observed standard deviations around the mean of the observed noise is contained within the interval of the three predicted standard deviations around zero (i.e., the mean of the predicted noise).

4.5 Swaptions

Swaptions is a financial analysis application that calculates the price of a portfolio of swaption financial instruments using Monte Carlo simulation. We present the swaptions computation in Figure 12. The loop is located in the function `HJM_Swaption_Blocking` in the file `HJM_Swaption_Blocking.cpp`, starting at line 157. The loop in this computation consumes almost 100% of the execution time. The loop performs multiple Monte Carlo trials, adding contributions of the individual iterations to the final price of the swaption. The accumulated value of the swaption is divided at the end of the computation by the number of trials.

```
float dPrice = 0.0;
for (i = 0; i <= lTrials - 1; i += blocksize) {
   float simres = runSimulation(this, i)
   dPrice += simRes;
}

double dMeanPrice = dPrice / lTrials;
```

Figure 12. Swaptions Computation

Mean Pattern Analysis. We present the results of the probabilistic analysis of the perforated computation for the variable `dMeanPrice` in Row 3 of the Table 3.

The inputs of the analysis are assumed to be independent and chosen from a normal distribution. As Figure 13 indicates, the observed input distribution is skewed to the right ($g = 0.73$). However, given that the predicted perforation noise greater than the mean observed perforation noise, the normal distribution approximation provides a conservative estimate of the mean perforation noise.

4.6 Streamcluster

Streamcluster is an unsupervised learning algorithm that discovers patterns in a stream of high-dimensional points. Streamcluster is designed to operate in small memory environments. Figure 14 presents the computation that we perforate. The computation is located in the function `pFL` in the file `streamcluster.cpp` at line 600. The computation finds an optimal set of cluster centers for points in working set and a summary of previously seen points. The function `pgain` calculates the improvement of the new set of candidate cluster centers. The algorithm stops if an improved solution was not found in `iter` steps. If the calculated improvement is greater than zero, the points are assigned to their new centers.

```
float change = 0.0;

for (i = 0; i < iter; i++) {
  x = i % numfeasible;
  change += pgain(candidates[x], points);
}
```

Figure 14. Streamcluster Computation

Sum Pattern Analysis. The outputs of the perforated program do not essentially depend on the loop induction variable — multiple sets of candidate cluster centers may yield a similar improvement in the clustering quality. This is reflected by small differences between the sum returned by the original computation and the sum returned by the perforated computation. The loop in its essence performs iterative refinement, where the contributions of each index do not depend on the skipped iterations. Extrapolation of the perforated result actually increases the difference between the two sums beyond the predicted value.

Figure 15 shows that the distribution of the inputs is highly asymmetric: it has one peak at the value zero, where about 98% of the probability mass is located, and a number of values greater than 5000. The skewness coefficient of this distribution is $g = 24.77$. We attribute the large mass located at a singular point 0 as a reason for the large standard deviation of the output value for this computation. Note that the mean output value of this computation does not change significantly after perforation. Extrapolating the

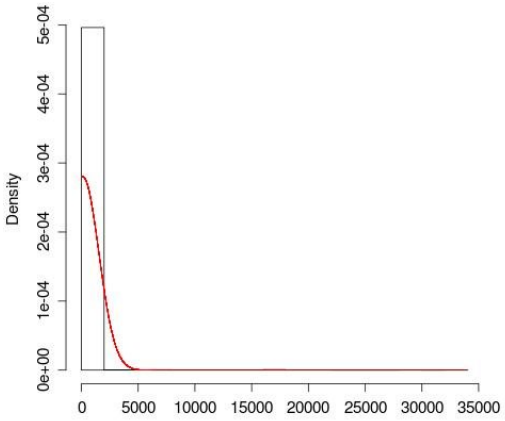

Figure 15. Histogram of values returned by the function `pgain` and fitted normal distribution from Sum analysis

output from the perforated computation therefore causes a large absolute perforation noise.

4.7 x264

x264 is a video encoder for the new H.264 high-definition video standard. It takes as input a raw video stream and generates a compressed video file as output. The encoder applies various heuristic algorithms to find parts of the video stream that are good candidates for compression, causing the final video to have acceptably good quality and a small size.

A major part of the video encoding is motion estimation — a process which searches previously encoded images for blocks of pixels that are similar to those in the image that it is currently encoding. If it finds sufficiently similar blocks, it can efficiently encode the current block as a delta that references the previously encoded block. The computation in Figure 16 assigns the dissimilarity scores between the `current` block and the list of neighboring `blocks`. In each step of the inner loop the program calculates the similarity between pixels at position `p` in function `ssd`. The inner loop sums the similarity contributions of all pixels.

The similarity score between blocks (`i_satd`) is compared against the best score seen so far, and the index of the best block is updated if the new sum value is the best seen so far. The computation terminates before exceeding all `qpel_iters` steps if it has not seen an improvement in last four steps. We perforate only the inner loop (with induction variable `x`) of this computation.

Argmin-Sum Analysis. The computation increments the variable `i_satd` in every iteration of the inner loop, with a new score from comparing pixels at position `p` calculated in the function `ssd`. The final value of `i_ssd` is used for comparison with the best similarity score seen so far. We can apply the *argmin-sum pattern* on the value of the `i_satd` variable and the values of the input terms computed by the function `ssd`. To calculate the mean and variance of the predicted perforation noise we obtain the average number of block comparisons, i.e. the number of iterations of the outer loop. We use the generated logs to find an average number of comparisons L, which is $L \approx 6$.

The analysis assumes a uniform distribution of the variables Z_i. We obtain the values of Z_i as a difference between the original and perforated sums. The uniform distribution gives equal probability of occurrence of smaller and larger perforation noise, with an

```
int bestidx = 0;
int bestsum = 0;

for( b = 4*qpel_iters; b > 0; i-- ) {

  i_satd = 0;

  // perforatable loop:
  for( p = 0; x < i_height * width; p += 4 ) {
    i_satd += ssd(current, blocks[index[b]], p, data)
  }

  if ( i_satd < bestsum ) {
    bestsum = i_satd;
    bestidx = index[b];
  }

  if( i%4 == 0 && early_stop(bestidx, bestsum, i_satd) )
    break;

}
```

Figure 16. x264 Computation

expected skewness coefficient equal to 0. The empirical distribution of the data, however, is right-skewed ($g = 2.99$). When estimating the parameters of the uniform distribution, we use the values that cover over 95% of the empirical probability mass. Since Z is the sum on $m - n$ terms, the bounds on 95% of the values vary with the perforation rate. Table 3 presents the bounds for all three perforation rates.

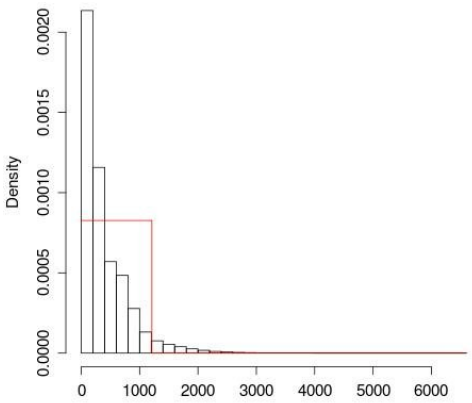

Figure 17. Histogram of values of variable Z_i and fitted uniform distribution from argmin-sum analysis

Table 3, Row 5 presents the results of the probabilistic analysis of the perforated computation. The predicted mean and variance of the perforation noise is considerably larger than the mean observed perforation noise. We attribute the difference to assumptions in the computation model. First, the encoded video data is not independent, since most of the adjacent pixels, and also pixels in neighboring blocks have similar intensities. Second, the uniform distribution assumption on the values of Z is a conservative estimate of the true distribution. This assumption assigns much higher scores to the blocks, thus increasing the predicted perforation noise.

Simulation-Based Bounds. We compare the empirical (observed) 99th percentile perforation noise with the estimates a) obtained through simulation, and b) calculated from the analytical

Perforation Rate	> 99% Bound		
	Empirical	Simulation	Analysis
0.25	105.0	539.1	2935
0.50	119.0	857.7	4220
0.75	197.0	1147	6218

Table 5. Argmin-Sum Simulation Numbers

expression derived from the Chebyshev's inequality. We present the perforation noise in Table 5. Column 1 presents the perforation rate. Column 2 presents the empirical perforation noise recorded during the program execution (also present in Table 4). Column 3 presents the perforation noise obtained by extrapolating the simulation results from Table 1. Column 4 presents the perforation noise bound obtained from Chebyshev's inequality (also present in Table 4). The noise in all cases is calculated for $L \approx 6$ and $n \approx 8$.

Both the bound obtained from the simulation, and the bound obtained from the analytical expression, are from 5 to 35 times larger than the empirically observed bound. The simulation bound is 4.2-3.3 times smaller than the analytical bound. It therefore provides a tighter estimate of the true empirically observed bound for this benchmark.

4.8 Worst Case Analyses

Table 6 presents the worst-case perforation noise described in Section 3.5. The first and the second columns present the name of the application and the analyzed pattern. Column 3 (Input Range) bounds the interval of input variables for the worst-case analysis. We repeat the same number we reported in Table 4. Column 4 (Perforation Rate) presents the perforation rate. Column 5 (Worst Case Noise) presents the result of the worst-case analysis. We use the number of loop iterations from Table 2 to calculate the worst case perforation noise. For all benchmarks except x264 the mean number of iterations is equal to the maximum number of iterations. For x264, we use the maximum number of iterations, 16, that the application executes to calculate the worst case perforation noise.

The remaining three columns compare the worst-case analysis results against the results we previously obtained using probabilistic and statistical techniques. Column 6 (vs Predicted Mean Noise) presents the ratio between the worst-case perforation noise and the corresponding predicted perforation noise from Table 3. Column 7 (vs Observed Mean Noise) presents the ratio between the worst-case perforation noise and the mean observed perforation noise from Table 3. Finally, Column 8 (vs 99% Predicted Bound) presents the ratio between the worst-case perforation noise and the predicted perforation noise at level 99% (i.e., a noise which is greater than at least 99% of the perforation noise) from Table 4.

Compared to the predicted and the observed mean perforation noise, the worst-case perforation noise is often an order of magnitude larger. The difference between the worst-case perforation noise and the predicted mean perforation noise ranges from 7 times (x264 argmin-sum and bodytrack sum pattern) to 91 times (swaptions mean pattern) larger. We do not report the ratio for the bodytrack sum division pattern because the predicted perforation noise is 0. A difference between the worst-case analysis and the observation from the program execution is even greater. The worst-case perforation noise is from 7 times (bodytrack sum-division) to 1000 times (swaptions) larger than the corresponding observed mean perforation noise.

To provide a further insight into the probability of occurrence of large perforation noise, we calculate ratio between the worst case perforation noise and the predicted bound on the absolute perforation noise at the level 99%. For three patterns (bodytrack sum, swaptions mean and streamcluster sum), the worst case perforation noise is greater, often by more than 50% than the perforation noise at the level 99%. For bodytrack sum-division and x264 argmin-sum the perforation noise at the level 99% is larger than the worst case perforation noise, indicating an overly conservative estimate. Those two patterns use looser Chebyshev-style bounds, which make fewer assumptions about the inputs. For all patterns, the worst case perforation noise itself is greater than the maximum observed noise.

Our worst-case analysis operates with a compressed range of values that excludes many observed values. Nevertheless, a comparison of the worst-cast analysis results with the observed perforation noise indicates that the worst-case analysis significantly overestimates the perforation noise from running applications. The results from the probabilistic analysis often provide tighter, more accurate, but still conservative bounds.

5. Probabilistic Acceptability Specifications

Probabilistic acceptability specifications provide the transformation system with the information it needs to apply rich transformations. Our acceptability specifications apply to individual blocks of code. Each block of code has inputs (variables that it reads whose values come from outside the block), outputs (variables or return values whose values are calculated inside the block), and specified expressions (expressions whose values the analysis abstracts with random variables).

5.1 Output Specifications

Each output specification identifies a variable or return value, then specifies a probabilistic constraint on that variable or return value. The constraints work either with the moments of the perforation noise or a probabilistic bound on the perforation noise. Each moment specification identifies a bound on the mean, variance, or other moment of the absolute perforation noise $|D|$ introduced by any transformation. A probabilistic bound on the absolute perforation noise $|D|$ specifies both a probability p and a bound B. The specification states that, with probability p, the absolute perforation noise $|D|$ should be less than B.

5.2 Input Specifications

Each input specification identifies a variable, then specifies a probability distribution for the values stored in that variable. There are several base probability distributions (for example, normal distributions with a specified mean and variance, uniform distributions with specified bounds, and histograms of recorded or otherwise obtained values). It is also possible to specify the distribution using computations over previously specified input variables or ghost variables. This is the mechanism enables the specification of correlated distributions such as random walks or ghost variables.

5.3 Expression Specifications

In the analysis of the block of code, the transformation system may encounter expressions that it should abstract as selected from a given probability distribution. Conceptually, each evaluation of the expression is modeled as another choice from the underlying probability distribution. To enable the construction of correlated distributions, the choice history is available to specify the value of the next choice.

5.4 Example

We next present an example that illustrates some of the intended functionality. Figure 18 presents a block of code with a probabilistic specification. At this point the probability distributions are fully specified, including parameters such as mean and variance. The distributions can either be directly specified by the developer or, in a previous pass, automatically inferred from values collected during representative executions of the program.

Application	Pattern	Input Range	Perforation Rate	Worst Case Noise	vs Predicted Mean Noise	vs Observed Mean Noise	vs $> 99\%$ Predicted Bound
bodytrack	sum	$[a, b] = [0.0, 20.0]$	0.25	200	8x	24x	1.4x
			0.50	400	7x	18x	1.9x
			0.75	600	8x	7.5x	2.4x
bodytrack	sum division	$[a, b] = [0, 20]$ $[c, d] = [1000, 1200]$	0.25	0.084	-	21x	0.6x
			0.50	0.142	-	20x	0.6x
			0.75	0.227	-	7x	0.6x
swaptions	mean	$[a, b] = [0.0, 0.2]$	0.25	0.05	71x	500x	18x
			0.50	0.1	91x	1000x	10x
			0.75	0.15	75x	750x	6x
streamcluster	sum	$[a, b] = [0, 12371]$	0.25	160823	33x	31x	1.6x
			0.50	309275	37x	33x	2.1x
			0.75	457727	32x	26x	2.6x
x264	argmin sum	$[a, b] = [0, 490.1]$	0.25	1960.4	7x	156x	0.7x
			0.50	3920.8	9x	267x	0.9x
			0.75	5881.2	9x	232x	0.9x

Table 6. Comparision of the Worst Case Perforation Noise against the Observed Deviations

```
block {
  inputs: a[i] : N(1, 1),
          g[i] : U(1, 2),
          b[i] = + { g[j] . 0 <= j && j < 20 };
  outputs: P(|s| > 2 * n) < 0.01,
           mean(|s|) < 2 * sqrt(n),
           var(|s|) < 3 * n;
} {
  s = 0;
  for (i = 0; i < n; i++) {
    expression: f(i) in N(2, 1);
    s = a[i] + 2*b[i] + 3*f(i);
  }
}
```

Figure 18. Example Probabilistic Specification

Inputs Clause: The `inputs` clause specifies that the analysis should model `a[i]` as an infinite sequence of random variables. The distribution is i.i.d. normal with mean 1 and variance 1. `g[i]` is an infinite sequence of random variables, i.i.d. uniform with lower bound 1 and upper bound 2. `g[i]` is a ghost variable — its only purpose is to specify the distribution for `b[i]`, which the analysis models as containing overlapping sums from `g[i]` with a window size of 20.

Outputs Clause: The outputs clause specifies that the probability of the absolute perforation noise `|s|` for the result `s` exceeding `2*n` must be less than 1%. It also specifies that mean of the absolute perforation noise `|s|` should be less than `2 * sqrt(n)` and the variance should be less than `3 * n`.

Expression Clause: The `expression` clause in the body of the computation specifies that the analysis should model the sequence of values for the `f(i)` expression as a sequence of i.i.d normal random variables with mean 2 and variance 1.

Analysis: Using techniques similar to those discussed in Section 1.1, the analysis can determine that the perforation noise for `s` with interleaved perforation at rate 0.5 is a normal random variable with mean 0 and variance 6`n` (assuming `n` is even). The absolute perforation noise `|s|` is therefore half-normal with mean $\sqrt{12\mathtt{n}/\pi}$ and variance $6\mathtt{n}(1 - 2/\pi)$. The key facts that the analysis must use are:

- **Means and Variances:** The mean of a sum of random variables is the sum of the means of the random variables. The mean of the difference of two random variables is the difference of the means.

  The variance of a sum or difference of independent random variables is the sum of the variances of the random variables.

  Multiplying a random variable with mean μ and variance σ^2 by a constant c scales the random variable to produce a new random variable with mean $c\mu$ and variance $c\sigma^2$.
- **Sums Of Uniform Random Variables:** The sum of uniform random variables quickly approximates a normally distributed random variable.
- **Sums of Normal Random Variables:** The sum of normal random variables is another normal random variable.

Because the perforation noise for `s` is normal, over 99.7% of its values lie within three standard deviations of the mean. The standard deviation of `s` is less than $2.5\sqrt{\mathtt{n}}$, so the probability `P(|s| > 7.5*sqrt(n)) < 99.7`.

Output Specification Satisfaction Check: The output specification for `|s|` requires that the probability `P(|s| > 2*n) < 0.01`. If `2*n > 7.5 * sqrt(n)`, the specification is satisfied. This inequality holds if `n > 16`. So the compiler will introduce a dynamic check on the value of `n`, choosing the original version when `n < 16` and the perforated version otherwise.

The mean of `|s|` is approximately `1.95 * sqrt(n)` and the variance of `|s|` is approximately `2.18 * n`. So the perforated computation always satisfies the mean and variance output specification for `|s|`.

Note that the analysis models a sum of `n` i.i.d. uniform random variables as a normal distribution. To ensure the accuracy of the model, the compiler must ensure that `n` is large enough to ensure the accuracy of the model. In general, we anticipate that requiring `n > 4` will ensure sufficient accuracy.

6. Automating The Perforation Transformation

The analysis presented in Section 3 establishes the theoretical foundation for the principled application of rich transformations. The empirical results presented in Section 4 show that the assumptions behind the theoretical analysis from Section 3 can be satisfied by real-world computations. Section 5 presents probabilistic acceptability specifications that developers can use to specify acceptable differences between the results that the original and transformed computations produce. The remaining step is to automate the ap-

plication of transformations on this foundation. We propose an automated transformation system that performs the following steps:

- **Pattern Recognition:** Each theoretical analysis from Section 3 characterizes the perforation noise for a specific computational pattern under specific assumptions about the distribution of the values on which the pattern operates. The first step is to build a static program analysis system that can recognize the analyzed patterns when they appear in the source code of the program. We expect that it will be possible to leverage existing analyses such as reduction recognition [7, 10] for this purpose. It should also be feasible to build new analyses designed specifically for this purpose.
- **Assumption-Based Analysis Selection:** The transformation system next selects a theoretical analysis to apply for the recognized pattern (or more generally, patterns). One approach selects the analysis based on how well its distributional assumptions match the observed values from executions of the program on representative inputs. The transformation system first automatically produces an instrumented version of the program that, when it runs, records the values on which the recognized patterns operate. The transformation system then runs this instrumented version on representative inputs to record the values on which the patterns operate.

  The next step is to use the recorded values to select the set of assumptions that best match the observed data. For each pattern, the transformation system will have at its disposal a variety of different analyses, each of which makes certain assumptions about the distribution of the data on which the pattern operates and each of which produces expressions that characterize the perforation noise under those assumptions. The transformation system will, for each analysis, fit the underlying distributions to the recorded values. It can then apply statistical tests to see if the fitted distributions (conservatively) match the recorded values, and if so, use the analysis with the best fit.

  An alternative to using empirical data is to provide the transformation system with distribution information (for example, from a type system) about the values on which the pattern operates. The transformation then uses the specified distribution information to drive the analysis selection.
- **Prediction-Based Analysis Selection:** Prediction-based analysis selection can be used either as an additional validation step for or as an alternative to the above assumption-based analysis selection. Here the decision is based not on how well the recorded input values match the distributional assumptions from the theoretical analysis. The decision is instead based on how well the theoretical analysis predicts the observed perforation noise from the instrumented runs. To obtain this prediction, the transformation system may, if necessary to obtain a quantitative prediction, first fit the underlying probability distributions from each theoretical analysis to the recorded input values. It then uses the perforation noise expressions to predict the actual perforation noise in the computation. A comparison selects the analysis that best (conservatively) predicts the observed noise.
- **Accuracy Specification:** At this point the transformation system has matched the pattern and found an analysis that can predict the consequences of perforating the pattern. The next step is to find a perforation strategy that optimizes some performance goal while probabilistically keeping the perforation noise acceptably low. We therefore assume the presence of an accuracy specification that specifies either 1) apply the perforation strategy that maximizes performance subject to keeping the perforation noise within specified bounds (typically specified as an absolute or relative difference between the results that the original and transformed patterns produce) or 2) apply the perforation strategy that minimizes perforation noise subject to meeting a specified performance goal (typically expressed as a reduction in the running time by a specified integer factor).
- **Transformation:** The transformation system next takes the accuracy specification and the expressions that characterize the perforation noise and and generates a corresponding mathematical programming problem whose solution determines the perforation factor `k` to apply to the pattern. It solves the problem, derives `k`, and appropriately perforates the loop.
- **Noise Propagation:** Perforatable computations often appear embedded within larger computations. To evaluate the effect of the perforation on the end to end result that the computation produces, the transformation system can use sampling to derive a model of how the computation responds to changes in the result that the perforated computation produces. Using multiple sampling runs, the transformation system systematically replaces the result from the original computation with different results to derive an empirical perforation response function for the application [14, 15]. A developer or user can then determine if the perforation is acceptable within the larger context of the complete application.

  It may also be possible to use a theoretical analysis or type system to characterize the effect of the perforation on the end to end application result.
- **Combining Transformations:** In some applications there may be multiple perforation opportunities. In such applications we anticipate sampling or searching the induced multiple pattern perforation space to find optimal points that perforate multiple patterns simultaneously [9, 14].

6.1 Simulation-Based Analysis

It is also possible to find potential perforation targets empirically — perforate each loop in turn, then observe the effect on the end to end result that the computation produces. This approach may find perforation opportunities that do not match any of the analyzed computational patterns. In this case the transformation system can use directed sampling/simulation to characterize the introduced perforation noise [16]. The transformation system would run the perforated pattern on randomly selected input values (potentially with the distribution obtained by fitting standard distributions to the values observed during executions of the program on representative inputs) and recording the resulting perforation noise to derive an empirical model of how the computation responds to perforation. This empirical model can then substitute for the theoretical model to drive the applied perforation strategy as described above.

6.2 Multiple Versions and Dynamic Adaptation

In addition to using perforation to produce a single optimized version of the application, perforation can also generate multiple versions of the application, each of which occupies a different point in the underlying performance versus accuracy trade-off space [9]. The theoretical analyses in this technical report can be used to understand the shape of the trade-off space and drive the dynamic selection of perforation policies that satisfy optimization targets that change as the computation executes.

6.3 General Classes of Computations

The basic analysis presented in this technical report generalizes to include with large classes of computations such as linear computations followed by non-linear operations such as minimum and division. It is also possible to generalize the proposed program analysis to recognize these general class of computations (as opposed to patterns). So, for example, any program analysis capable of recogniz-

ing general linear functions could be combined with our probabilistic reasoning approach to enable rich transformations with probabilistic accuracy guarantees.

6.4 Map Reduce

Many distributed computations can be formulated as a combination of the basic programming language constructs map and reduce — a map applies a computation to a set of values; a reduce then combines the mapped values to obtain the final result. Formulating the computation in this way directly exposes the structure to the static perforation analysis. An analysis of the reduce operator can characterize the effect of perforating the computation (the perforated computation would simply apply the map reduce computation to a subset of the original set of values). One advantage of this approach is the simplicity of the model of computation, in which the pattern is explicit and therefore directly available for analysis and transformation without the need for program analysis.

7. Related Work

Rinard uses linear regression to obtain probabilistic timing and accuracy models for computations that skip tasks [14, 15]. These models characterize the complete application, not subcomputations. Task skipping is similar to loop perforation in that many of the skipped tasks correspond to blocks of loop iterations.

Rinard et. al. propose the use of Monte-Carlo simulation to explore how loop perforation changes the result that computational patterns produce [16]. In addition to these empirical Monte-Carlo techniques, this technical report proposes probabilistic reasoning to derive symbolic expressions that characterize this effect.

Ramsey and Pfeffer define a stochastic lambda calculus [12] for probabilistic modeling — the programs work directly with probability distributions. Our techniques, in contrast, work with deterministic programs. We use probability distributions to model uncertainty about the values the program manipulates and probabilistic reasoning to justify the application of transformations that produce a transformed program that (deterministically) produces different results from the original program.

Reed and Pierce present a type system for capturing function sensitivity, which measures how much a function may magnify changes to its inputs [13]. The type system uses deterministic worst-case reasoning. Chaudhury, Gulwani, and Lublinerman present a program analysis for automatically determining if a function is continuous or not [5]. Once again, the reasoning is deterministic and worst-case. In contrast, we use probabilistic reasoning, not worst-case reasoning, to obtain probabilistic bounds, not worst-case deterministic bounds, on how program transformations, not small changes in the input, change the results that the program computes.

A purely empirical application of loop perforation can enable the system to explore the underlying performance versus accuracy tradeoff that loop perforation induces [9]. This empirical evaluation operates at the level of the complete application (as opposed to subcomputations). A dynamic control system can use the resulting characterization to maximize accuracy while meeting real-time responsiveness requirements on platforms with fluctuating performance characteristics.

Dynamic Knobs [8] converts static application configuration parameters into dynamic control variables. The system can then use these control variables to change the point in the underlying performance versus accuracy space at which the application executes. One control strategy maximizes accuracy subject to satisfying real-time responsiveness requirements on platforms with fluctuating performance characteristics.

Petabricks [1], Green [3], and Eon [18] all allow developers to provide multiple implementations of a specific piece of application functionality, with different implementations exhibiting different performance versus accuracy tradeoffs. The system can then select an implementation that best meets its current needs. There is no explicit reasoning to justify the acceptability of the different alternatives — the system relies on the developer to specify only acceptable alternatives.

8. 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 and statistical reasoning justify the application of transformations that may, within probabilistic bounds, change the result that the program produces. The goal is to provide a more accurate 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 and statistical reasoning to bound how much loop perforation may change result that the program produces. This reasoning can provide the foundation required to understand, predict, and therefore justify the application of loop perforation. We anticipate the development of similar approaches for other rich transformations.

References

[1] J. Ansel, C. Chan, Y. L. Wong, M. Olszewski, Q. Zhao, A. Edelman, and S. Amarasinghe. Petabricks: A language and compiler for algorithmic choice. In *ACM SIGPLAN Conference on Programming Language Design and Implementation*, Dublin, Ireland, Jun 2009.

[2] A. Asuncion and D. Newman. UCI machine learning repository, 2007.

[3] W. Baek and T. Chilimbi. Green: A framework for supporting energy-conscious programming using controlled approximation. In *ACM SIGPLAN Conference on Programming Language Design and Implementation*, June 2010.

[4] C. Bienia, S. Kumar, J. P. Singh, and K. Li. The PARSEC benchmark suite: Characterization and architectural implications. In *PACT-2008: Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, Oct 2008.

[5] S. Chaudhuri, S. Gulwani, and R. Lublinerman. Continuity analysis of programs. In *POPL*, pages 57–70, 2010.

[6] P. C. Diniz and M. C. Rinard. Dynamic feedback: An effective technique for adaptive computing. In *PLDI*, pages 71–84, 1997.

[7] M. Hall, B. Murphy, S. Amarasinghe, S. Liao, and M. Lam. Interprocedural analysis for parallelization. *Languages and Compilers for Parallel Computing*, pages 61–80, 1996.

[8] H. Hoffman, S. Sidiroglou, M. Carbin, S. Misailovic, A. Agarwal, and M. Rinard. Dynamic knobs for power-aware computing. In *ASPLOS '11*, 2011.

[9] 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, MIT, Sept. 2009.

[10] K. Kennedy and J. R. Allen. *Optimizing compilers for modern architectures: a dependence-based approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2002.

[11] S. Misailovic, S. Sidiroglou, H. Hoffmann, and M. Rinard. Quality of service profiling. In *ICSE*, 2010.

[12] N. Ramsey and A. Pfeffer. Stochastic lambda calculus and monads of probability distributions. In *POPL*, pages 154–165, 2002.

[13] J. Reed and B. C. Pierce. Distance makes the types grow stronger: a calculus for differential privacy. In *ICFP*, pages 157–168, 2010.

[14] M. Rinard. Probabilistic accuracy bounds for fault-tolerant computations that discard tasks. In *Proceedings of the 20th annual international conference on Supercomputing*, pages 324–334. ACM New York, NY, USA, 2006.

[15] M. Rinard. Using early phase termination to eliminate load imbalancess at barrier synchronization points. In *OOPSLA 2007*, Montreal, Oct. 2007.

[16] M. C. Rinard, H. Hoffmann, S. Misailovic, and S. Sidiroglou. Patterns and statistical analysis for understanding reduced resource computing. In *Onward!*, 2010.

[17] R. Rubinfeld. Sublinear time algorithms. In *Procedings of International Congress of Mathematicians*, 2006.

[18] 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*.