Probabilistic Accuracy Bounds for Perforated Programs: A New Foundation for Program Analysis and Transformation

Citation: Martin Rinard. 2011. Probabilistic accuracy bounds for perforated programs: a new foundation for program analysis and transformation. In Proceedings of the 20th ACM SIGPLAN workshop on Partial evaluation and program manipulation (PEPM '11). ACM, New York, NY, USA, 79-80.

As Published: http://dx.doi.org/10.1145/1929501.1929517

Publisher: Association for Computing Machinery (ACM)

Persistent URL: http://hdl.handle.net/1721.1/72443

Version: Author's final manuscript: final author's manuscript post peer review, without publisher's formatting or copy editing

Probabilistic Accuracy Bounds for Perforated Programs

A New Foundation for Program Analysis and Transformation

Martin Rinard

Massachusetts Institute of Technology

rinard@mit.edu

Categories and Subject Descriptors F.3.2 [*Semantics of Programming Languages*]: Program Analysis; I.2.2 [*Automatic Programming*]: Program Transformation; D.3.4 [*Processors*]: Optimization

General Terms Performance, Reliability, Accuracy, Verification, Program Analysis, Program Transformation

Keywords Loop Perforation, Program Analysis

1. Motivation

Traditional program transformations operate under the onerous constraint that they must preserve the exact behavior of the transformed program. But many programs are designed to produce approximate results. Lossy video encoders, for example, are designed to give up perfect fidelity in return for faster encoding and smaller encoded videos [10]. Machine learning algorithms usually work with probabilistic models that capture some, but not all, aspects of phenomena that are difficult (if not impossible) to model with complete accuracy [2]. Monte-Carlo computations use random simulation to deliver inherently approximate solutions to complex systems of equations that are, in many cases, computationally infeasible to solve exactly [5].

For programs that perform such computations, preserving the exact semantics simply misses the point. The underlying problems that these computations solve typically exhibit an inherent performance versus accuracy trade-off — the more computational resources (such as time or energy) one is willing to spend, the more accurate the result one may be able to obtain. Conversely, the less accurate a result one is willing to accept, the less resources one may need to expend to obtain that result. Any specific program occupies but a single point in this rich trade-off space. Preserving the exact semantics of this program abandons the other points, many of which may be, depending on the context, more desirable than the original point that the program happens to implement.

2. Loop Perforation

Loop perforation can automatically generate variants of a given computation, with the variants occupying a rich set of points spread across the underlying perfomance versus accuracy trade-off space [3, 6–8]. Systems can build on the availability of this range of different computations to, for example, produce acceptably accurate results more quickly or with less energy [3, 7, 8], dynamically adapt to changes in the underlying computational platform [3], eliminate barrier idling [7], identify promising opportunities for manual optimization [6], or survive failures [3, 8].

2.1 Loop Perforation Transformation

Loop perforation transforms loops to perform a subset of their original iterations. For example, given the following loop:

```
for (i = 0; i < n; i++) { sum += a[i]; }
```

loop perforation can produce the following transformed loop:

```
for (i = 0; i < n; i += k) { sum += a[i]; }
sum = sum * k;
```

The transformed loop simply executes every `k`th iteration of the original loop, then extrapolates `sum` to obtain an approximation to the result that the original loop produces (of course, other policies such as skipping an initial or final block of iterations or skipping a psuedorandomly selected set of iterations are also possible). For many such loops, perforation produces a computation that executes in significantly less time (for our set of benchmark programs chosen from the PARSEC benchmark suite [1] typically between three to five times faster than the original computation) with modest accuracy losses (the result differs by less than 10% from the result that the original program produces).

3. Probabilistic Reasoning

How can one justify the application of loop perforation? We model the effect of perforation on the above loop by modeling the values `a[i]` as independent random samples from the same probability distribution with mean μ and finite variance σ^2. The expected value of `sum` is $\mathtt{n}\mu$ for both the original and perforated loops. The probability that the absolute difference between the two values of `sum` is greater than d is less than or equal to $(\mathtt{n}(\mathtt{k}-1)\sigma^2)/d^2$ for d at least $\sigma\sqrt{\mathtt{n}(\mathtt{k}-1)}$. If we further assume that the `a[i]` are drawn from a Gaussian distribution, the expected value of the absolute difference between the two values of `sum` is $\sigma\sqrt{(k-1)2\mathtt{n}/\pi}$.

If adjacent values of `a[i]` are positively correlated (this can be the case, for example, if they come from a spatially sampled domain such as an image or geographic data whose adjacent values tend to change slowly or if they come from computations on overlapping sets of numbers), it is possible to improve the estimate of the expected difference.

For example, assume the `a[i]` are a Gaussian random walk with step size variance σ^2 (i.e., the `a[i]` are a Markov chain where `a[i+1]-a[i]` is Gaussian distributed with mean 0 and variance σ^2). For `k=2`, the expected absolute difference between the two values of `sum` is $\sigma\sqrt{\mathtt{n}/\pi}$. If we drop the Gaussian assumption and sim-

PEPM'11, January 24–25, 2011, Austin, Texas, USA.
ACM 978-1-4503-0485-6/11/01.

ply assume independent increments between adjacent `a[i]` with finite variance σ^2, the probability that the absolute difference between the two values of `sum` is greater than d is less than $\mathtt{n}\sigma^2/2d^2$ for d at least $\sigma\sqrt{\mathtt{n}/2}$.

It is often possible to generalize such analyses to model arbitrary linear combations of the input data followed by nonlinear operations such as `min`, `max`, division, and multiplication. It is possible to validate such analyses by running the program on representative inputs and either performing statistical tests on the values that the program manipulates, observing how well the analytic model (conservatively) predicts the observed differences in the results that the original and perforated computations produce, or both. In the absence of an accurate analytic model of the computation (this can happen if the computation is too complex to model analytically), simulation may provide an effective way to model the effect of perforation on the accuracy of the computation [9].

4. A New Foundation

This probabilistic approach provides a new foundation for program analysis and transformation. Traditional analyses use discrete logic to reason about the behavior of the program and justify transformations that are guaranteed to preserve the exact semantics. We instead propose analyses that probabilistically bound the accuracy loss that the transformation may introduce. Potentially in combination with desirable-accuracy specifications, such probabilistic analyses provide the justification required to transform the computation in a principled way with guarantees about the effect of the transformation on the result that the computation produces. To the best of our knowledge, this is a fundamentally new paradigm for program analysis and transformation and the first to deliver program transformations that change the result that the program produces in predictable ways with analytically justified guarantees.

5. Building on the Foundation

So how can one use this new foundation? A first step is to identify patterns that interact in guaranteed ways with accuracy-preserving transformations such as loop perforation [9]. Systems can then recognize such patterns as they occur in the program and transform them appropriately. It is possible to generalize this approach to programs that compute arbitrary linear functions (potentially composed with nonlinear operations at the end of the computation) Profiling may help the program transformation system identify time-consuming computations that are appropriate optimization targets.

Probabilistic reasoning can also justify new transformations such as approximate memoization (returning previously computed values for function invocations that are close in time, parameter values, and/or the function itself to previous invocations) or targeted linear thinning (approximating or eliminating computations of terms in linear expressions with small coefficients).

6. Propagating Transformed Results

In many cases transformed computations are invoked by larger client applications. A common scenario, for example, is the use of loop perforation to obtain more efficient heuristic search metrics [3]. When it is possible to trace the effects through the client to the result that the application produces, analytic models can accurately predict the global consequences of applying the transformation. When the client is too complex to model analytically, we propose the use of empirical propagation analysis (systematically exploring the effect of changing the result that the transformed computation returns back to the client) to predict the global effect of the transformation and justify the transformation of selected subcomputations.

7. Potential Uses

As with traditional optimizations, it is possible to apply the transformation policy (for example, the choice of loop performation factors `k`) statically before the program runs. It is also usually straightforward to generate code that can dynamically change the transformation policy. This capability enables the construction of systems that measure various aspects of the interaction of the program with the underlying computational platform, then dynamically adapt the transformation policy to realize goals such as eliminating idling at barriers placed at the end of parallel loops [7], reducing the fixed cost of provisioning for peak load in clusters of servers [4], or maximizing accuracy while meeting real-time deadlines in the face of phenomena (for example, voltage scaling, processor failures, or load fluctations) that cause fluctuations in the amount of computational resources that the underlying computing platform delivers to the program [3].

8. Conclusion

Since the inception of the field, researchers developing analyses and transformations have operated under the onerous constraint of preserving the exact program semantics. Given the broad (and increasing) range of programs designed to perform inherently approximate computations, this constraint is no longer appropriate. We present a fundamentally new paradigm that is appropriate for such computations. This paradigm uses probabilistic analysis to justify transformations that change, within guaranteed probabilistic accuracy bounds, the result that the transformed program produces.

Acknowledgments

Dan Roy performed the analysis presented in Section 3. Hank Hoffman, Sasa Misailovic, Stelios Sidiroglou, and Anant Agarwal all contributed to various aspects of the loop perforation project.

References

[1] 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.

[2] J. Hartigan and M. Wong. A k-means clustering algorithm. *JR Stat. Soc., Ser. C*, 28:100–108, 1979.

[3] H. Hoffman, S. Misailovic, S. Sidiroglou, A. Agarwal, and M. Rinard. Using Code Perforation to Improve Performance, Reduce Energy Consumption, and Respond to Failures. Technical Report TR-2009-042, Computer Science and Artificial Intelligence Laboratory, MIT, Sept. 2009.

[4] H. Hoffmann, S. Sidiroglou, M. Carbin, S. Misailovic, A. Agarwal, and M. Rinard. Power-Aware Computing with Dynamic Knobs. Technical Report TR-2010-027, MIT CSAIL, May 2010.

[5] M. Kalos and P. Whitlock. *Monte carlo methods*. Wiley-VCH, 2008. ISBN 352740760X.

[6] S. Misailovic, S. Sidiroglou, H. Hoffmann, and M. Rinard. Quality of service profiling. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1*. ACM, 2010.

[7] M. Rinard. Using early phase termination to eliminate load imbalancess at barrier synchronization points. In *Proceedings of the 2007 ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages, and Applications*, Oct. 2007.

[8] M. Rinard. Probabilistic accuracy bounds for fault-tolerant computations that discard tasks. In *Proceedings of the 20th Annual International Conference on Supercomputing*, 2006.

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

[10] x264. http://www.videolan.org/x264.html.