Using Bayesian Optimization for Hardware Design

Orianna DeMasi
Computer Science Division
University of California, Berkeley
odemasi@eecs.berkeley.edu

Joseph Gonzalez
Computer Science Division
University of California, Berkeley
jegonzal@eecs.berkeley.edu

Benjamin Recht
Computer Science Division
University of California, Berkeley
brecht@berkeley.edu

James Demmel
Computer Science Division and Mathematics Department
University of California, Berkeley
demmel@berkeley.edu

Abstract

The problem of searching complex spaces with difficult to model or even black-box functions arises in many domains. One such domain is hardware design in computer architecture. As hardware and software become more sophisticated, the importance and difficulty of finding an optimal design increases. We explore the use of Bayesian optimization on a problem of hardware design. Applying this approach on an example where traditional methods to optimize blackbox functions don't apply, this method outperforms a naive approach in both number of samples and the quality of the solution found.

1 Introduction

The need to efficiently optimize expensive black box response functions arises in many applications in system tuning. Examples of these applications include algorithm hyper parameter tuning [1, 2, 3, 4], automatic performance tuning of software [5, 6, 7, 8], and hardware design [9, 10, 11, 12]. Often in these applications search spaces are far too large to exhaustively enumerate. Even worse, in many cases these tuning spaces contain discrete dimensions and complex combinatorial constraints. Finally these spaces will only grow in size and complexity as computer software and hardware continue to become more sophisticated and heterogenous.

In this work we explore a model based approach to system tuning in the setting of hardware design. We apply previous work in model based optimization to address the unique challenges of the complex combinatorial structure presented in the context of computer processor design.As with prior work in Baysian optimization, we simultaneously estimate approximation $\hat{f}(x)$ of the unknown black box function $f(x)$ as well as the optimizing configuration x^* by evaluating f (i.e., simulating a processor design) on a sequence of configurations $(x_1, \ldots, x_n)$. Following previous work, we assume Gaussian process priors and find they model the space well.

To balance the exploration of the overall space as well as the micro-optimization of particularly successful hardware configurations, we adopt a Bayesian approach to minimize the number of configurations that must be evaluated. We estimate the uncertainty in the function $\hat{f}$ and apply both an upper confidence bounds selection rule and expected improvement selection rule to determine the sequence of configurations to evaluate.

Using real processor simulation data we demonstrate the applicability and impressive performance of Bayesian optimization. We show that by simultaneously estimating $\hat{f}$ and balancing exploration and exploitation of the hardware design space we can substantially reduce the number of processor

simulations required to achieve a near optimal configuration. In the context of hardware design this can make a substantial difference in the overall system performance and enable the design of more complex processors.

2 Background

Computer processor design presents three major challenges to system optimization. First, measuring performance of potential designs is extremely expensive due to cycle-accurate simulations being many orders of magnitude slower than the final hardware implementations that they simulate. Simulations of even small benchmarks will take hours or even days for a single configuration and multiple benchmarks must be measured to assess one configuration. Second, design spaces consist of millions of points. Exhaustive exploration of these spaces is infeasible. The design spaces are discrete and many rules impose constraints; not all configurations are valid. Finally, even tiny improvements are necessary, as they have the opportunity to have a large impact over the lifespan of the system.

Much work has been done to make individual simulations cheaper by only simulating sample periods of a benchmark or using regression techniques to make statistical approximations to the performance function $f(x)$ [10, 11, 9]. Another response is to reduce the number of observations needed by using classical search methods or using domain understanding to guide the search [12].

We propose using Bayesian optimization, which combines global modeling, to prevent termination at a local optimum, with active sampling to focus on areas needing more sampling. Our work suggests this is an especially useful approach in that it does not rely on deep domain expertise, so it is applicable to a variety of applications.

3 Method

As in classical optimization settings, the goal is to optimize a response $f(x)$ over a set of configurations $x \in X$ where the space X is a bounded set of possible parameter configurations. We use a surrogate model $\hat{f}(x|(x_i, y_i)_{i=1}^n)$, which is built on the previously observed n points $x_i, i = 1...n$ and their measured responses $y_i, i = 1...n$. The model provides both an approximation to the performance at any point and a measure of uncertainty. Ideally, we would find the optimal configuration $x^* = \arg\max_{x \in X} f(x)$. In practice, we approximate this point by searching for $\hat{x}^* = \arg\max_{x \in X} \hat{f}(x|(x_i, y_i)_{i=1}^n)$. It is important to note that our goal is not to build a model that minimizes modeling error over the entire space, but to guide a search towards the optimal point x^*. In this general framework for Bayesian optimization [13] we are left to decide which prior over functions and which acquisition function to use. In this work, we consider a Gaussian process prior and both expected improvement and upper confidence bounds for the acquisition function.

3.1 Model - Gaussian processes

Gaussian processes [14] are a popular prior to assume over functions [1, 2, 13]. Their generality, elegance, and uncertainty measure are very attractive in addition to the ability to write down many closed form expressions for calculations. More formally, a Gaussian process is a collection of random variables such that any finite subset has a multivariate distribution. A Gaussian process is fully defined by its predictive mean and covariance functions and a deeper discussion of their properties can be found in other resources [14].

3.2 Acquisition function - Expected improvement and upper confidence bounds

There are a variety of popular acquisition functions, most notably, the probability of improvement [15], expected improvement (EI), and, more recently, upper confidence bounds (UCB) [16]. Here we consider expected improvement and upper confidence bounds.

First, for Gaussian processes, the expected improvement can be written out in closed form as

$$E[I(x)] = (\mu(x) - f_{max})\Phi\left(\frac{\mu(x) - f_{max}}{\sigma(x)}\right) + \sigma(x)\phi\left(\frac{\mu(x) - f_{max}}{\sigma(x)}\right)$$

where f_{max} is the maximum value observed, $\Phi(x)$ is the cumulative distribution function, and $\phi(x)$ the density function of the standard normal. Further, $\mu(x)$ is the returned prediction from the model and $\sigma(x)$ is the uncertainty of the model. In this setting, the next point to sample x^{next} is the one that maximizes the expected improvement

$$x^{next} = \arg\max_{x \in X} E[I(x)]$$

This method was developed for minimizing a function, so here, where we are maximizing an objective, we minimized the negative value of the objective value.

We also consider the idea of exploiting upper confidence bounds. With this approach, we choose the next point to sample by

$$x^{next} = \arg\max_{x \in X} \{\mu(x) + \sigma(x)\}.$$

4 Experimental Setup and Dataset

We consider optimizing the performance of micro architectural processor design on a suite of benchmarks by maximizing the performance of the proposed hardware, as measured in billions of instructions per second (bips). We consider both optimizing for individual benchmarks, as well as the full suite by optimizing the average performance across benchmarks. Various points in the design space have been verified by running simulations. In this experiment, we are constrained to use publicly available data[1], which may now be searched exhaustively at low cost. However, our experiment shows that collecting just a small subset of this very expensive data would suffice for optimization. To test each search method, we perform 25 searches randomly initializing the search at a different point each time. The average performance across these runs (and error bars of standard deviation when appropriate) is reported.

The open source dataset contains the performance of 2,000 microarchitectural configurations of a proposed computer processor [11]. It was previously used in the context of developing regression models for processor design spaces [10]. The performance of each configuration is reported for 7 SPEC2k benchmarks (ammp, apple, equake, gcc, gzip, mesa, twolf) and for SPECjbb, a Java server benchmark. The performance was established via a simulation of a 100 million instruction subsample of the benchmark in Turandot, a parameterized out-of-order superscalar processor simulator [17] . The static dataset provides an example of a heavily constrained search space. The parameters of the dataset include pipeline width and depth, number of physical registers, branch reservations and latency, and the size and latencies of the instruction and data caches. Further description of the design space can be found in previous work [10, 11].

5 Empirical Analysis

The results of the search methods are presented in figure 1. The horizontal axis is the number of points that were sampled, or the expense of the search, and the vertical axis is the performance of the best performing configuration that has been tested thus far, or the success of the searches. We include randomly sampling points from the space, as random search does surprisingly well in many applications and is the naïve approach. Other classical blackbox optimization methods, including Nelder Mead and genetic methods, are difficult to apply in this heavily constrained arena, so they are omitted here due to lack of a fair comparison.

We see in figure 1 that all the Bayesian methods do very well. In a matter of less than 30 observations they all get remarkably close to finding the optimal point in the dataset or reach it. The figure on the right has error bars indicating one standard deviation from the average search performance. The error bars are very tight, indicating that all 25 performed searches achieved very good results and the average performance is representative of individual searches. Of note is how poor random sampling does in this application. Here random sampling continues to improve as more of the space is explored, but the figure has been truncated to highlight the Bayesian methods.

We tried a few variations of the searches, for example changing the correlation kernel from squared exponential to linear [14] and varying the autocorrelation value. The search was fairly robust to

[1] `http://people.duke.edu/~bcl15/software.html`

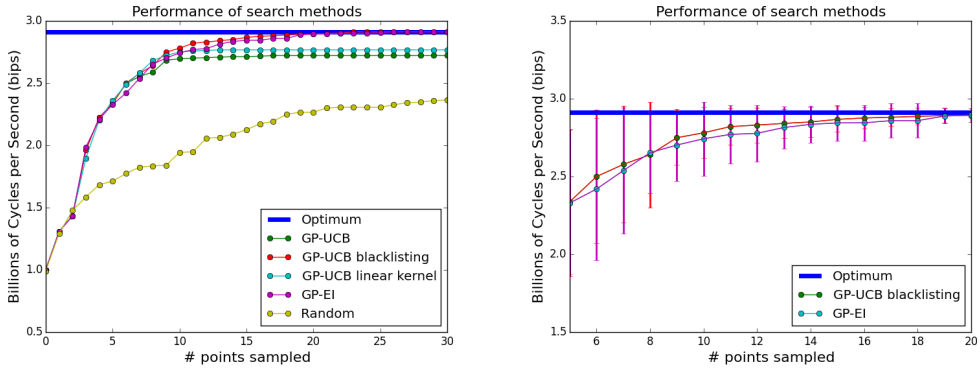

Figure 1: Performance of search methods with different acquisition functions on average bips across benchmarks compared with the naive approach. The optimum is the known best solution in the space and the error bars in the figure on the right indicate standard deviation between trials.

these changes, but there was a significant effect from changing the acquisition function. Using a direct implementation of UCB the search would get stuck and stop exploring the space. This could be useful for implementing a stopping heuristic, but as can be seen in figure 1 using UCB the search would get stuck on a suboptimal point. To get around this, we tried blacklisting observed configurations and forcing UCB to always sample a new point. This solved the problem and UCB was then able to reach the optimal solution at the same rate as using the EI criterion.

6 Discussion

There are many applications where the search space is not a perfect hypercube in parameter space, but instead a warped domain, e.g. unions of hypercubes. These situations arise frequently in complier optimization and computer architecture where the setting of one parameter dictates the existence or range of other parameters. While our constraints arose from the nature of the dataset, they present an example of one of these heavily constrained datasets, which the method had no problem handling, but traditional blackbox optimization methods would have struggled with.

In this example, we looked at optimizing the performance for individual benchmarks and presented the results of optimizing for a general best performing processor by maximizing the average bips across benchmarks. The flexible and generalizable approach used permits domain experts to readily assign alternative, task-appropriate weights to benchmarks. The performance optimizing individual benchmarks was similar, but often converged to different solutions, implying that different configurations are better for different compute kernels, as expected.

7 Conclusion

In this work we have proposed using Bayesian optimization with a Gaussian process prior for hardware design and studied the performance on a dataset of computer processor microarchitecture optimization. The Bayesian approach is very general and was able to be applied in the heavily constrained space that we were working in, where other methods did not have direct applicability. The search approach performed well by finding a better quality of solution and in fewer function evaluations than random search.

Acknowledgments

We would like to extend perfuse thanks to Benjamin Lee for generously making his data available to the public [10, 11] and for his advice. We would also like to thank Krste Asanović for his suggestions, as well as Christopher Celio, Adam Izraelevitz, and Yunsup Lee for valuable discussions.

References

[1] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems*, pages 2951–2959, 2012.

[2] James S Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems*, pages 2546–2554, 2011.

[3] Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *Learning and Intelligent Optimization*, pages 507–523. Springer, 2011.

[4] Frank Hutter, Holger Hoos, and Kevin Leyton-Brown. An evaluation of sequential model-based optimization for expensive blackbox functions. In *Proceeding of the fifteenth annual conference companion on Genetic and evolutionary computation conference companion*, pages 1209–1216. ACM, 2013.

[5] Matteo Frigo and Steven G Johnson. Fftw: An adaptive software architecture for the fft. In *Acoustics, Speech and Signal Processing, 1998. Proceedings of the 1998 IEEE International Conference on*, volume 3, pages 1381–1384. IEEE, 1998.

[6] Markus Püschel, José MF Moura, Bryan Singer, Jianxin Xiong, Jeremy Johnson, David Padua, Manuela Veloso, and Robert W Johnson. Spiral: A generator for platform-adapted libraries of signal processing alogorithms. *International Journal of High Performance Computing Applications*, 18(1):21–45, 2004.

[7] Jeff Bilmes, Krste Asanovic, Chee-Whye Chin, and Jim Demmel. Optimizing matrix multiply using phipac: a portable, high-performance, ansi c coding methodology. In *Proceedings of the 11th international conference on Supercomputing*, pages 340–347. ACM, 1997.

[8] R Clint Whaley and Jack J Dongarra. Automatically tuned linear algebra software. In *Proceedings of the 1998 ACM/IEEE conference on Supercomputing*, pages 1–27. IEEE Computer Society, 1998.

[9] Engin Ipek, Sally A McKee, Karan Singh, Rich Caruana, Bronis R de Supinski, and Martin Schulz. Efficient architectural design space exploration via predictive modeling. *ACM Transactions on Architecture and Code Optimization (TACO)*, 4(4):1, 2008.

[10] Benjamin C Lee and David M Brooks. Accurate and efficient regression modeling for microarchitectural performance and power prediction. In *ACM SIGPLAN Notices*, volume 41, pages 185–194. ACM, 2006.

[11] Benjamin C Lee and David M Brooks. A tutorial in spatial sampling and regression strategies for microarchitectural analysis. *IEEE Micro Special Issue on Hot Tutorials*, 27(3):74–93, 2007.

[12] Sandeep Navada, Niket K Choudhary, and Eric Rotenberg. Criticality-driven superscalar design space exploration. In *Proceedings of the 19th international conference on Parallel architectures and compilation techniques*, pages 261–272. ACM, 2010.

[13] Eric Brochu, Vlad M. Cora, and Nando de Freitas. A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *CoRR*, abs/1012.2599, 2010.

[14] Carl Edward Rasmussen. Gaussian processes for machine learning. 2006.

[15] Harold J Kushner. A new method of locating the maximum point of an arbitrary multipeak curve in the presence of noise. *Journal of Fluids Engineering*, 86(1):97–106, 1964.

[16] Niranjan Srinivas, Andreas Krause, Sham M Kakade, and Matthias Seeger. Gaussian process optimization in the bandit setting: No regret and experimental design. *arXiv preprint arXiv:0912.3995*, 2009.

[17] Mayan Moudgill, J-D Wellman, and Jaime H Moreno. Environment for powerpc microarchitecture exploration. *Micro, IEEE*, 19(3):15–25, 1999.