Intelligent Heuristic Construction with Active Learning

CGO: G: Intelligent Heuristic Construction with Active Learning

William F. Ogilvie
University of Edinburgh
s0198982@sms.ed.ac.uk

Pavlos Petoumenous
University of Edinburgh
ppetoume@inf.ed.ac.uk

Zheng Wang
Lancaster University
z.wang@lancaster.ac.uk

Hugh Leather
University of Edinburgh
hleather@inf.ed.ac.uk

ABSTRACT

Building effective optimization heuristics is a challenging task which often takes developers several months if not years to complete. Predictive modelling has recently emerged as a promising solution, automatically constructing heuristics from training data, however, obtaining this data can take months per platform. This is becoming an ever more critical problem as the pace of change in architecture increases. Indeed, if no solution is found we shall be left with out of date heuristics which cannot extract the best performance from modern machines.

In this work, we present a low-cost predictive modelling approach for automatic heuristic construction which significantly reduces this training overhead. Typically in supervised learning the training instances are randomly selected, regardless of how much useful information is present. Our approach, on the other hand, uses *active learning* to carefully select the more informative examples, thus reducing the training time.

We demonstrate this technique by automatically creating a model to determine on which device to execute four parallel programs at differing problem dimensions for a representative CPU–GPU based system. Our methodology is remarkably simple and yet effective, making it a strong candidate for wide adoption. At high levels of classification accuracy the average learning speed-up is 3x, as compared to the state-of-the-art.

1. INTRODUCTION

Creating analytical models on which optimization heuristics can be based has become harder as processor complexity has increased. Compiler developers often have to spend months if not years to get a model perfected for a single target architecture. Many modern compilers support a wide range of disparate platforms, and since heuristics are not portable within or out-with processor families this has ultimately resulted in out of date compilers [20].

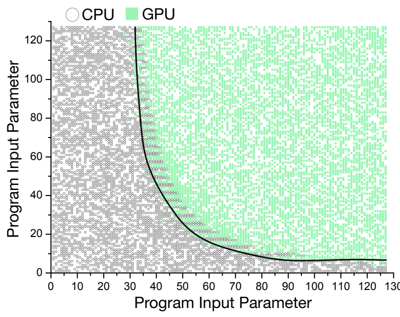
Machine Learning based predictive modelling has rapidly emerged as a viable means of automating heuristic construction [13, 30]; by running example programs (optimized in different ways) and observing how the variations affect program run-time a machine learning tool can predict good settings with which to compile new, as yet unseen, programs. This new research area is promising, having the potential to fundamentally change the way compiler heuristics are designed, but suffers from a number of issues. One major concern is the cost of collecting training examples. That is to say, while machine learning allows us to automatically construct heuristics with little human involvement, the cost of generating training examples (that allow a learning algorithm to accumulate knowledge) is often very expensive.

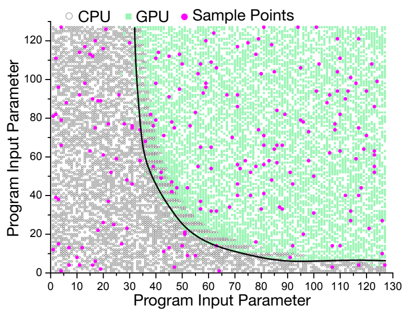
In this work we present a novel, low-cost predictive modelling approach that can significantly reduce the overhead of collecting training examples for parallel program mapping without sacrificing prediction accuracy. The usual procedure in supervised learning is to passively collect training examples at random, regardless of how useful they might be to the learner. We propose using *active learning* instead, which is a method by which the learning algorithm itself is able to iteratively choose the training instances it believes carry the greatest information based upon whatever knowledge it has already accumulated. Specifically, we use active learning to automatically construct a heuristic to determine which processor will give the better performance on a CPU–GPU based heterogeneous system at differing problem sizes for a given program. We evaluate our system by comparing it with the typical random sampling methodology, used in the bulk of prior work. The experimental results show that our technique accelerates training by a factor of 3x on average.

2. MOTIVATION

To motivate our work, we demonstrate how much unnecessary effort is involved in the traditional random-sampling based learning techniques, and point out the extent to which a better strategy can improve matters. In Fig. 1(a) we show for `HotSpot`, from the Rodinia [5, 6] suite, when it is better to run on the CPU *versus* the GPU for maximum performance. The benchmark accepts two independent program inputs, and their values form the axes of the graph. The graph data itself was generated by randomly selecting 12,000 input combinations and running them on both the CPU and GPU enough times to make a statistically sound decision about which device is better for each, where a boundary line approximately separates the regions at which either device should be chosen.

In Fig. 1(b) a random selection of 200 inputs to `HotSpot` is chosen, as might be typical in a standard 'passive' learning technique. From this data a heuristic is created with the RandomCommittee machine learning algorithm from the Weka tool-kit [17], and the heuristic achieves a respectable 95% accuracy. Machine learning can clearly learn good


(a) The problem space


(b) Random sample points

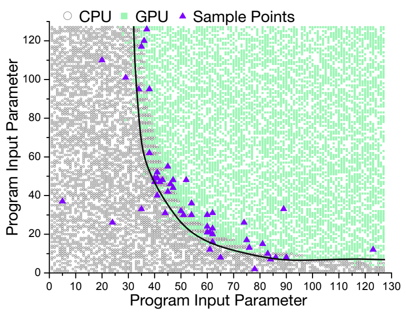

(c) Intelligent sample points

Figure 1: Figure (a) shows the problem space of the Rodinia `HotSpot` benchmark. 12,000, 2-dimensional program inputs are run to discover which device (Cpu or Gpu) gives the better performance. A boundary line approximately separates the parts of the space where Cpu and Gpu are better. Figure (b) shows a random selection of 200 inputs. Using RandomCommittee to learn a heuristic with these inputs achieves an accuracy of 95%. Figure (c) shows an intelligent selection of 31 inputs. Using RandomCommittee to learn a heuristic with these inputs achieves an accuracy of 97%, representing a 6x speed-up in training time.

heuristics, but our intuition insists that the majority of the randomly selected inputs offer little useful information.

We prove this intuition in Fig. 1(c) where we have instead selected just 31 inputs and once again asked the RandomCommittee algorithm to learn a heuristic. Using fewer than 15% as many observations as the standard passive learning technique we achieve an accuracy of 97%. There is, therefore, significant potential to reduce the training cost for the machine learned heuristics if we could only choose the right inputs to train over.

3. OUR APPROACH

As a case study, this work aims to train a predictor to determine the best processor to use for a given program input. We wish to avoid profiling inputs that provide little or no information for the learning algorithm to train over. We achieve this by using active learning which carefully chooses each input to be profiled in turn, based on what is already known.

Figure 2 provides an overview of our approach. First, some number of program inputs are chosen at random to 'seed' the algorithm and these are then profiled to determine the better device for them – Cpu or Gpu. What follows is a number of steps which progressively add to the set of training examples until some termination criterion are met. To select which program input to add to the training set for profiling, a number of different, intermediate models are created using the current training set and different machine learning algorithms. Our method then searches for an input for which the intermediate models or heuristics most disagree on whether it should be run on the Cpu or the Gpu. The intuition is that the more these models agree on an input, the less likely it will improve the prediction accuracy of the learned heuristic.

The technique for choosing new training inputs is called *Query by Committee* (Qbc) [27] and is described in Sect. 3.1, Sect. 3.2 explains our disagreement metric, whilst Sect. 3.3 details how the inputs are profiled: particularly, how the decision about whether the input should be run on the Cpu or on the Gpu is made statistically sound.

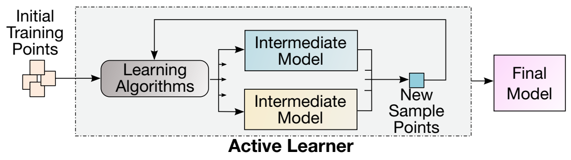


Figure 2: An overview of our active learning approach. Initially, we use a few random samples to construct several intermediate models. Those models are utilized to choose which new data point is to be profiled next. The new sampled data point is then used to update the models. We repeat this process until a certain termination criterion is met where a final model will be produced as the outcome.

3.1 Query by Committee

The key idea behind active learning is that a machine learning algorithm can perform better with fewer training examples if it is allowed to choose the data from which it learns. There are a number of approaches available [4] but we employ a heterogeneous implementation of *Query by Committee* (Qbc), a widely utilized active learning technique.

The 'committee' in our Qbc implementation consists of a number of distinct learning algorithms that are initially trained with a small set of randomly collected examples. In our case, each training example is a set of program inputs with a label indicating which processor gives better performance when these inputs are profiled. As these models are initially built from a small set of examples they are unlikely to be highly accurate at first, but we are able to iteratively improve these models with the following steps using new, carefully chosen, training examples.

The challenge in active learning lies in how to select those training examples that are most likely to improve the prediction accuracy of a model out of all the potential examples we could use. In Qbc this is achieved by asking each distinct model to make predictions on a random collection of program inputs not currently present in the training set - called the *candidate set*. Since each model is built using a different algorithm they may or may not reach consensus as to whether a particular program input should be run on a particular device. We then only profile those inputs for which the 'committee' *disagrees* the most, and then add those new examples into the training set. The learning loop begins again by creating its distinct intermediate models using the new information it has gathered and carefully selects another informative example to learn from. This procedure repeats until some completion criterion are met.

The insight behind QBC is that we do not want to create new training instances from parts of the problem-space which are already collectively understood by the committee of algorithms.

3.2 Assessing Disagreement

We use information entropy (1) [9] to evaluate the level of disagreement for each potential candidate that could be added to the training set next, where $p(x_i)$ is the proportion of committee members that predict that the instance (set of inputs) x is fastest on device i of n. From within all the candidates found to have the maximum entropy value in a given iteration of the learning loop one example is randomly chosen to be profiled next. This means the inputs associated with that instance are run on the CPU and GPU kernels and it is determined which processor is faster under those conditions. The relevant label is given to the instance and the information added to the training set. The learning loop begins another iteration and the intermediate models formed again, but this time the new information is taken into account.

$$H(x) = -\sum_{i=1}^{n} p(x_i) \log p(x_i) \qquad (1)$$

3.3 Statistically Sound Profiling

Since computer timings are inherently noisy we use statistics to increase the reliability of our models. In particular, we record a minimum number of timings from each device, as specified by the user. We use Interquartile Range [24] outlier removal then apply Welch's t-test [33] to discover if one hardware device is indeed faster than the other. If we cannot conclude from the t-test that this is the case, then we perform an equivalence test. Both devices are said to be 'equivalent' if the difference between the higher mean plus its 95% confidence interval minus the lower mean minus its confidence is within some threshold of indifference. In our system this threshold was set to be within 1% of the minimum of the two means. If the fastest device cannot be determined and they are not equivalent an extra set of observations are obtained and the tests applied again, up until some user defined number of tries. In the case of equivalence or the threshold of attempts being reached the CPU is chosen as the preferred device since it is more energy-efficient.

4. EXPERIMENTAL SETUP

We evaluated our approach on a heterogeneous platform comprised of an Intel Core i7 4770 (3.4 GHz) CPU and NVIDIA GeForce GTX Titan (6GB) GPU. The benchmarks we used were taken from the Rodinia suite, namely `HotSpot`, `PathFinder`, and `SRAD`. We also included a simple matrix multiplication application. Each benchmark was chosen because it has equivalent OPENMP and OPENCL implementations and a multidimensional problem space.

Benchmark	Dim	Min	Max	Step	Size	Cand
HotSpot	2	1	128	1	16,384	10,000
MatMul	3	1	256	1	$1.6x10^7$	10,000
Pathfinder	2	2	1024	1	$1.0x10^6$	10,000
SRAD	2	128	1024	16	3,136	2,636

Table 1: The sizes of the input spaces for each benchmark. *Dim* indicates the number of dimensions – each dimension is then treated in the same way for our case study. *Min* gives the minimum value of each dimension. *Max* gives the maximum value of each dimension. *Step* gives the step value on each dimension. *Size* is the total number of points in the input space. *Cand* is the number of points in the candidate set for each benchmark.

QBC *Algorithms.*

Our committee comprised 12 unique machine learning algorithms, taken from the Weka tool-kit. They are `Logistic`, `MultilayerPerceptron`, `IB1`, `IBk`, `KStar`, `RandomForest`, `LogitBoost`, `MultiClassClassifier`, `RandomCommittee`, `NNge`, `ADTree`, and `RandomTree`: all configured with default parameters. These 12 were specifically selected for their capability of producing a binary predictor from numeric inputs and the fact that they have been widely used in prior work.

Program Input space.

The sizes of the input spaces for each benchmark were chosen to give realistic inputs to learn over. Table 1 describes the number of dimensions and their range for each benchmark. That table also gives the total number of inputs for each benchmark.

Initial Training Set and Candidate Set Sizes.

For all benchmarks the minimum initial training set size of one was used. The candidate set size (shown in Table 1) was chosen to be 10,000 examples, or the largest number possible under the problem size constraints.

Termination Criterion.

In all cases, the active learning iterations were halted at 200 steps. This value was selected because for all benchmarks the learning improvement had plateaued by that time.

Run-time Measurement and Device Comparison.

To determine if a benchmark is better suited to the CPU or GPU for a given input it was run on each processor at least 10 times, and at most 200 times. We employed a number of statistical techniques to ensure our data was as accurate as possible, including Welch's t-test, equivalence testing, and interquartile-range outlier removal.

Testing.

For testing, a set of 500 inputs were excluded from any training sets. Both our active and passive learning experiments were run 10 times and the arithmetic mean of the accuracy was taken, where the accuracy of each experiment was the average accuracy of all 12 models.

5. EXPERIMENTAL RESULTS

In this section we present the overall results of our experiments, showing that our active learning approach can

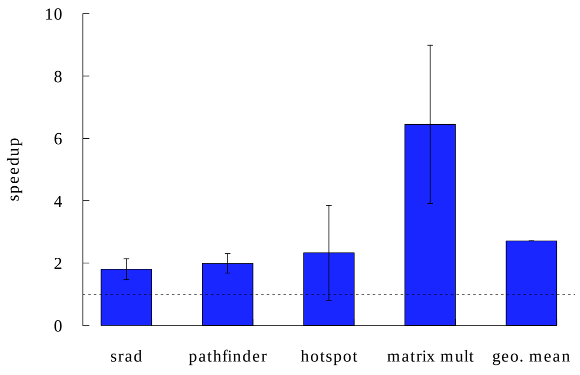


Figure 3: On average our methodology requires 3x fewer training examples to create a high quality heuristic than the traditional random-sampling technique, proving that this simple algorithm can save weeks, and potentially months, of compute time.

significantly reduce the training time by a factor of 3 when compared to the random sampling technique.

Figure 3 shows the average learning speed-up of our approach over the passive, random-sampling technique traditionally used in heuristic construction. The speed-up values are based on the number of inputs which need to be profiled in order to train a predictor to an accuracy of at least 85%. As can be seen from this figure, our approach constantly outperforms the classical random-sampling technique for all benchmarks, which in real terms means a saving of weeks to train these heuristics.

Moreover, if we look at the points selected by the Qbc algorithm for `HotSpot` in Fig. 4, we see they are reminiscent of the intelligent selection shown in Fig. 1(c) previously. In particular, the Qbc algorithm concentrates around the boundary line between devices, the most informative region in the space where the curve can be precisely predicted.

6. RELATED WORK

Analytic Modelling.

Analytic models have been widely used to tackle complex optimization problems, such as auto-parallelization [3, 25], runtime estimation [7, 34, 18], and task mappings [19]. A particular problem with them, however, is the model has to be re-tuned whenever it is targeted at new hardware [28].

Predictive Modeling.

Predictive modeling has been shown to be useful in the optimization of both sequential and parallel programs [8, 29, 31, 16]. Its great advantage is that it can adapt to changing platforms as it has no *a priori* assumptions about their behaviour but it is expensive to train. There are many studies showing it outperforms human based approaches [15, 21, 10, 35, 14, 11, 32]. Prior work for machine learning in compilers, as exemplified by the MilePost GCC project [12], often uses random sampling or exhaustive search to collect training examples. The process of collecting training examples could be expensive, taking several weeks if not months. Using active learning, our approach can significantly reduce the overhead of collecting training examples. This accelerates the process of tuning optimization heuristics using machine learning. The Qilin compiler [23] uses runtime profiling to predict a parallel program's execution time and map work across the CPU and GPU accordingly; however, where their approach is designed to work after deployment ours can be performed ahead of time, thereby reducing any slow-downs experienced by the end-user.

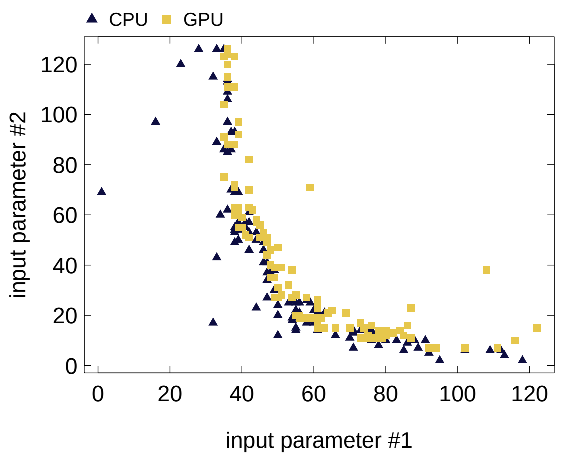


Figure 4: This graph shows the points selected by the Qbc algorithm to train on during one run to produce a heuristic for `HotSpot`. The inputs selected for training cluster around the boundary between device classes. These are the most informative points in the space since they precisely define the curve shown in Fig. 1(a).

Active Learning for Systems Optimization.

A recent paper by Zuluaga *et al.* [36] proposed an active learning algorithm to select parameters in a multi-objective problem. Their work is not concerned with single-objective workload scheduling and does not consider statistical soundness of raw data. Balaprakash *et al.* [1, 2] used active learning to reduce execution time of scientific codes but they only consider code variants and OpenCl parameters as inputs, they do not discuss the impact of problem size on performance. Furthermore, neither of these authors employed the Qbc methodology in their work.

Problem Size Optimization.

Optimizing code for different problem sizes in heterogeneous systems is discussed by Liu *et al.* [22] where they give an implementation of a compiler which uses a combination of regression trees and representative Gpu kernels, but their approach uses exhaustive search. Adaptic is a compilation system for Gpus [26] and uses analytical models to map an input stream onto the Gpu at runtime but their technique is not easily portable, where ours tackles that problem directly by making learning cheaper.

7. CONCLUSIONS

We have presented a novel, low-cost predictive modelling approach for machine learning based automatic heuristic construction. Instead of building heuristics based on randomly chosen training examples we use active learning to focus on those instances that improve the quality of the resultant models the most. Using QBC to construct a heuristic to predict which processor to use for a given program input our approach speeds up training by a factor of 3x, saving weeks of compute time.

8. REFERENCES

[1] P. Balaprakash, R. B. Gramacy, and S. M. Wild. Active-Learning-Based Surrogate Models for Empirical Performance Tuning. In *Proc. CLUSTER'13*.

[2] P. Balaprakash, K. Rupp, A. Mametjanov, R. B. Gramacy, P. D. Hovland, and S. M. Wild. Empirical Performance Modeling of GPU Kernels Using Active Learning. In *Proc. ParCo'13*.

[3] C. Bastoul. Code Generation in the Polyhedral Model Is Easier Than You Think. In *Proc. PACT'04*.

[4] C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., 2006.

[5] S. Che, M. Boyer, J. Meng, D. Tarjan, J. Sheaffer, S.-H. Lee, and K. Skadron. Rodinia: A Benchmark Suite for Heterogeneous Computing. In *Proc. IISWC'09*.

[6] S. Che, J. Sheaffer, M. Boyer, L. Szafaryn, L. Wang, and K. Skadron. A Characterization of the Rodinia Benchmark Suite with Comparison to Contemporary CMP Workloads. In *Proc. IISWC'10*.

[7] M. Clement and M. Quinn. Analytical Performance Prediction on Multicomputers. In *Proc. SC'93*.

[8] K. D. Cooper, P. J. Schielke, and D. Subramanian. Optimizing for Reduced Code Space using Genetic Algorithms. In *Proc. LCTES'99*.

[9] I. Dagan and S. P. Engelson. Committee-Based Sampling For Training Probabilistic Classifiers. In *Proc. ICML'95*.

[10] C. Dubach, T. Jones, E. Bonilla, G. Fursin, and M. F. P. O'Boyle. Portable Compiler Optimisation Across Embedded Programs and Microarchitectures using Machine Learning. In *Proc. MICRO'09*.

[11] M. K. Emani, Z. Wang, and M. F. P. O'Boyle. Smart, adaptive mapping of parallelism in the presence of external workload. In *CGO '13*.

[12] G. Fursin, C. Miranda, O. Temam, M. Namolaru, E. Yom-Tov, A. Zaks, B. Mendelson, E. Bonilla, J. Thomson, H. Leather, C. Williams, M. O'Boyle, P. Barnard, E. Ashton, E. Courtois, and F. Bodin. In *Proceedings of the GCC Developers' Summit*.

[13] D. Grewe, Z. Wang, and M. O'Boyle. Portable mapping of data parallel programs to opencl for heterogeneous systems. In *Proc. CGO '13*.

[14] D. Grewe, Z. Wang, and M. O'Boyle. Portable mapping of data parallel programs to opencl for heterogeneous systems. In *CGO '13*.

[15] D. Grewe, Z. Wang, and M. O'Boyle. A workload-aware mapping approach for data-parallel programs. In *HiPEAC '11*.

[16] D. Grewe, Z. Wang, and M. F. O'Boyle. Opencl task partitioning in the presence of gpu contention. In *LCPC '13*.

[17] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The WEKA Data Mining Software: An Update. *SIGKDD Explorations*, 2009.

[18] S. Hong and H. Kim. An Analytical Model for a GPU Architecture with Memory-level and Thread–level Parallelism Awareness. In *Proc. ISCA'09*.

[19] A. H. Hormati, Y. Choi, M. Kudlur, R. Rabbah, T. Mudge, and S. Mahlke. Flextream: Adaptive Compilation of Streaming Applications for Heterogeneous Architectures. In *Proc. PACT'09*.

[20] S. Kulkarni and J. Cavazos. Mitigating the compiler optimization phase-ordering problem using machine learning. In *Proc. OOPSLA '12*.

[21] S. Kulkarni and J. Cavazos. Mitigating the Compiler Optimization Phase-Ordering Problem using Machine Learning. In *Proc. OOPSLA'12*.

[22] Y. Liu, E. Z. Zhang, and X. Shen. A Cross-Input Adaptive Framework for GPU Program Optimizations. In *Proc. IPDPS'09*.

[23] C.-k. Luk, S. Hong, and H. Kim. Qilin: Exploiting Parallelism on Heterogeneous Multiprocessors with Adaptive Mapping. In *Proc. MICRO'09*.

[24] D. S. Moore and G. P. McCabe. *Introduction to the Practice of Statistics*. W. H. Freeman, 2002.

[25] L.-N. Pouchet, C. Bastoul, A. Cohen, and J. Cavazos. Iterative Optimization in the Polyhedral Model: Part II, Multidimensional Time. In *Proc. PLDI'08*.

[26] M. Samadi, A. Hormati, M. Mehrara, J. Lee, and S. Mahlke. Adaptive Input-aware Compilation for Graphics Engines. In *Proc. PLDI'12*.

[27] H. S. Seung, M. Opper, and H. Sompolinsky. Query by Committee. In *Proc. COLT'92*.

[28] M. Stephenson, S. Amarasinghe, M. Martin, and U.-M. O'Reilly. Meta Optimization: Improving Compiler Heuristics with Machine Learning. In *Proc. PLDI'03*.

[29] Z. Wang and M. F. O'Boyle. Mapping Parallelism to Multi-cores: A Machine Learning Based Approach. In *Proc. PPoPP'09*.

[30] Z. Wang and M. F. O'Boyle. Partitioning streaming parallelism for multi-cores: A machine learning based approach. In *Proc. PACT '10*.

[31] Z. Wang and M. F. O'Boyle. Partitioning streaming parallelism for multi-cores: a machine learning based approach. In *PACT '10*.

[32] Z. Wang and M. F. P. O'Boyle. Using machine learning to partition streaming programs. *ACM TACO*, 2013.

[33] B. L. Welch. The Generalization of "Student's" Problem when Several Different Population Variances are Involved. *Biometrika*, 1947.

[34] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, and P. Stenström. The worst-case execution-time problem – overview of methods and survey of tools. *ACM TECS*, 2008.

[35] M. Zuluaga, A. Krause, P. Milder, and M. Püschel. "Smart" Design Space Sampling to Predict Pareto–Optimal Solutions. In *Proc. LCTES'12*.

[36] M. Zuluaga, A. Krause, G. Sergent, and M. Püschel. Active Learning for Multi–Objective Optimization. In *Proc. ICML'13)*.