Active Learning Accelerated Automatic Heuristic Construction for Parallel Program Mapping

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

Pavlos Petoumenos
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 Indeed, if no solution is found we shall be left with out of date heuristics which cannot extract the best performance from modern machines.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 to evaluate regardless of how much useful information they carry, but this wastes effort on parts of the space that contribute little to the quality of the produced heuristic. Our approach, on the other hand, uses active learning to select and only focus on the most useful training examples and thus reduces the training overhead.

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.

Categories and Subject Descriptors

D.3.4 [**Programming Languages**]: Processors—*compilers, optimization*

Keywords

Active Learning; Machine Learning; Compilers

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage, and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s). Copyright is held by the author/owner(s).
PACT'14, August 24–27, 2014, Edmonton, AB, Canada.
ACM 978-1-4503-2809-8/14/08.
http://dx.doi.org/10.1145/2628071.2628128.

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, and since modern compilers often support a wide range of disparate platforms most have been found to be out of date [2].

Machine Learning based predictive modelling has rapidly emerged as a viable means of automating heuristic construction [1, 3]; 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. 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. OUR APPROACH

We use a heterogeneous Query-by-Committee (QBC) implementation of active learning, which is so-named because it requires that a number of different models be generated us-

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.

ing distinct member algorithms. The idea is that each Qbc member is given random training examples or instances. Since every member is different they form unique models from this information. To choose the next instance to learn from each algorithm in turn attempts to guess the classification (Cpu or Gpu) of pre-selected random unlabelled points. A subset is formed from these candidates comprising those instances for which the committee is in most disagreement. Finally, a single instance is chosen randomly from this group, it is then labelled and added to the training set of each classifier. This process repeats until some completion criterion are met.

3. 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.

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.

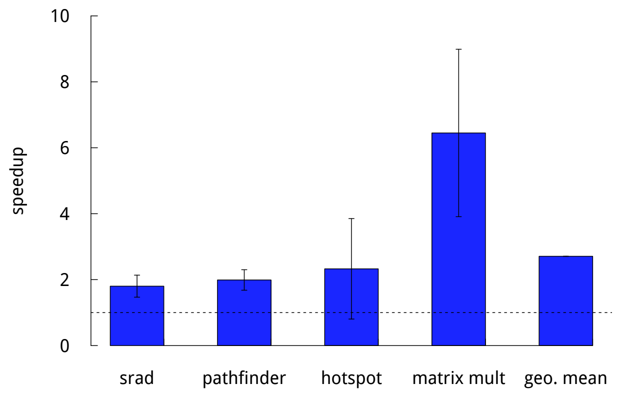

Figure 1: On average our approach requires 3x fewer training examples to create a high quality heuristic than a random sampling technique. These speed-up values are defined as the number of training instances required for the accuracy of a heuristic to plateau using the random technique divided by that for our Qbc implementation.

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.

4. RESULTS

Figure 1 shows the average learning speed-up of our approach over the classical random sampling technique. The actual values are based on the number of training examples required to produce a heuristic which is at least 90% accurate. Overall, we are able to accelerate training by 3x on average. In real terms, this translates to a saving of weeks of compute time.

5. REFERENCES

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

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

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