THE UNIVERSITY OF CHICAGO

CROSS ARCHITECTURE PERFORMANCE PREDICTION

A DISSERTATION SUBMITTED TO
THE FACULTY OF THE DIVISION OF THE PHYSICAL SCIENCES
IN CANDIDACY FOR THE DEGREE OF
MASTER OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

BY
YULIANA ZAMORA

CHICAGO, ILLINOIS
FALL 2019
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIST OF FIGURES</td>
<td>iv</td>
</tr>
<tr>
<td>LIST OF TABLES</td>
<td>v</td>
</tr>
<tr>
<td>ABSTRACT</td>
<td>vi</td>
</tr>
<tr>
<td>1 INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>2 MOTIVATION FOR PERFORMANCE PREDICTION</td>
<td>3</td>
</tr>
<tr>
<td>3 RELATED WORK</td>
<td>4</td>
</tr>
<tr>
<td>4 BACKGROUND</td>
<td>6</td>
</tr>
<tr>
<td>4.1 GPU Architectures: Nvidia P100 and V100</td>
<td>6</td>
</tr>
<tr>
<td>4.2 Architecture Performance: IPC</td>
<td>6</td>
</tr>
<tr>
<td>4.3 Active learning</td>
<td>8</td>
</tr>
<tr>
<td>4.4 Deephyper, NAS, and Balsam</td>
<td>9</td>
</tr>
<tr>
<td>5 INTRA-ARCHITECTURE IPC PREDICTION</td>
<td>11</td>
</tr>
<tr>
<td>6 INTER-ARCHITECTURE PREDICTION</td>
<td>12</td>
</tr>
<tr>
<td>6.1 Cross architecture memory bound prediction</td>
<td>12</td>
</tr>
<tr>
<td>6.2 Two stage Framework for IPC prediction</td>
<td>14</td>
</tr>
<tr>
<td>6.3 Neural architecture search at scale</td>
<td>15</td>
</tr>
<tr>
<td>7 EXPERIMENTAL RESULTS</td>
<td>19</td>
</tr>
<tr>
<td>8 DISCUSSION</td>
<td>29</td>
</tr>
<tr>
<td>8.1 Intra-architecture performance</td>
<td>29</td>
</tr>
<tr>
<td>8.2 Memory bound cross-architecture prediction</td>
<td>29</td>
</tr>
<tr>
<td>8.3 Inter-architecture IPC prediction</td>
<td>29</td>
</tr>
<tr>
<td>9 FUTURE WORK AND CONCLUSION</td>
<td>32</td>
</tr>
<tr>
<td>9.1 Future Work</td>
<td>32</td>
</tr>
<tr>
<td>9.2 Conclusion</td>
<td>32</td>
</tr>
</tbody>
</table>
LIST OF FIGURES

4.1 Comparison of P100 and V100 IPC .................................................. 8
6.1 Dram read and write utilization on both P100 and V100 GPUs ............... 12
6.2 Memory bound applications vs IPC of application on both P100 and V100 architectures .......................................................... 13
6.3 Normalized application percentage breakdown of data points that were chosen at random ................................................................. 15
6.4 Application data distribution of data points that were chosen at random ... 16
6.5 Normalized application percentage breakdown of data points created using active learning ............................................................... 17
6.6 Application data distribution of data points created using active learning ... 18
7.1 P100 IPC prediction using 451 training data points ............................ 19
7.2 P100 IPC prediction using 4,521 data points ..................................... 20
7.3 MAPE of each framework across applications tested ............................ 21
7.4 IPC prediction results of DeepHyper model using a training set with randomly chosen data points ..................................................... 21
7.5 IPC prediction results of DeepHyper model using a active learning curated training set ................................................................. 22
7.6 Prediction of backprop IPC using DeepHyper model with active learning chosen training set ............................................................. 23
7.7 Prediction of Stream application using model returned Deephyper using a training set with randomly chose data points ......................... 24
7.8 Prediction of Stream application using model returned Deephyper using a training set curated by the active learning model ....................... 25
7.9 Prediction of Stream application using model returned DeepHyper with random selection .............................................................. 26
7.10 Conventional deep learning architecture layout. ................................ 27
7.11 DeepHyper + Active learning architecture layout. ............................ 28
# LIST OF TABLES

4.1 Key specifications of selected GPUs of different generations ........................................ 7

7.1 Confusion matrix showing results of memory bound random forest classifier that predicts whether an application will be memory bound going from the P100 to V100 GPU architecture. ......................................................... 20
ABSTRACT

Given the rapid evolution of modern computing architectures, the time required to understand, benchmark, and model each new chip design can be daunting. Application performance depends on both the underlying hardware, as well as the efficiency of the application software stack. Since overall efficiency can depend strongly on hardware-specific software optimizations, early performance predictions can enable valuable software modifications before the target architecture is even in production. In this work, we explore a completely data-driven approach to hardware performance prediction. We leverage leadership-scale super-computing resources to build and train empirical models, with limited profiling data, without modifying any original source code. We employ both neural-architecture search (NAS) optimizations and active learning to automatically design and train deep learning models for the prediction of IPC (instructions per cycle) across two separate NVIDIA GPU architectures (P100 to V100). Although the massively-parallel approach certainly yields improved performance over simple empirical models, the accuracy of IPC predictions still falls short of expected real-world requirements.
CHAPTER 1
INTRODUCTION

In recent years, there has been significant investment in the development of new and innovative computing architectures that diverge from the general-purpose CPU. These new architectures include, among many others, graphical processing units (GPUs), neuromorphic computing chips, and field-programmable gate arrays (FPGAs). Much of this investment has been a direct response to a growing interest in the application of deep learning methodologies within both industry and scientific research. The future success of the software supporting these applications is strongly dependent on how efficiently it will run on future hardware. Understanding how to realize these gains can be a laborious process that delays the time to solution. Direct simulations of computing architectures can offer a clear trade-off between accuracy and performance, with cycle-accurate simulations being prohibitive for most real-world applications. On top of this, the development time needed to understand, benchmark, and model each new architecture can be extensive. This is especially problematic today, as the success of machine learning has sparked investment in a large number of new GPU-like architectures for compute-intensive neural-network training.

Since the ultimate performance of a software application depends so strongly on hardware-specific optimizations, especially for specialized architectures like GPUs, a large body of work has already aimed to build reliable performance prediction frameworks. The most-obvious motivation for such a framework is in the design of the new hardware itself. For example, the engineers designing the upgraded version of an existing GPU will certainly want to make sure that they don’t introduce any new performance regressions for well-known linear-algebra operations. A reliable performance-prediction framework allows the engineers to run these tests before any prototypes have been manufactured. Such a framework is also valuable for the evaluation of architectures that have left the design stage, but not readily available for testing. For example, the developers of an established deep-learning platform may want to
avoid certain hardware-specific optimizations if the modifications will not apply to the next generation of much faster chips.
CHAPTER 2

MOTIVATION FOR PERFORMANCE PREDICTION

Architecture benchmarking, port application to new hardware, and application profiling take a substantial amount of resources, advanced skills, and time. Additionally, scientists don’t want to make changes, whether small or insignificant to their applications. We want to make use of the current state of the art and in place systems to predict performance on future systems without the the time needed to do the tedious benchmarking. In the end, the development of a framework that allows for no code changes, will contribute to the understanding of performance of applications on these quickly developing and advancing hardware architectures.
CHAPTER 3

RELATED WORK

Architecture Performance Prediction

In line with our motivation, (1) want to understand the benefit of estimating GPU performance prior to writing a specific GPU implementation. Here, the authors look at going between a CPU implementation and GPU implementation by looking at corresponding counterparts. Their results show a predicted speedup going from CPU to GPU implementation.

(2) looks at relative performance between two platforms while only needing to observe short partial executions. Their method targets performance predictions in guidance for resource allocation. The partial executions require an API where the user must understand where the repetitive phases occur to understand execution behavior across the entire application without the need for full execution. The predictions and evaluations were done across CPU’s only and partial execution on the target is required in order to extrapolate and predict whole application performance.

(3) Created a toolkit that semi-automatically measures and models static and dynamic characteristics of applications using their binaries to predict the L1, L2, TLB cache miss counts, and execution time. Evaluations were also done on CPU only.

(4) Created a modeling framework, built on top of GROPHECY, that predicts kernel execution time and data transfer time to represent the total execution time of a GPU application.

(5) Created a GPU performance projection framework that estimates the performance benefit of using a GPU without requiring GPU programming but by providing pieces of CPU code that targets for GPU acceleration. These code skeletons are transformed to mimic tuned GPU codes where the cost and benefit of GPU development can then be estimated according to this transformed skeleton.
(6) Created a power and performance prediction model that predicts the optimal number of active processors for a given application.

(7) Created an easy to use model using one parallel application, SMG2000, to predict performance across two platforms. (8) Looked deeply into parameter space characterization for highly parameterized parallel applications.

(9) proposed a performance prediction procedure that has a set of kernel features extracted through an automated
CHAPTER 4
BACKGROUND

4.1 GPU Architectures: Nvidia P100 and V100

Though at face value, Nvidia’s P100 and V100 may not look very different, especially compared to older models, we can see substantial differences when comparing performance across these two architectures. Table 1 shows the architectural details of present and past Nvidia GPUs. It is clear that the last two columns are very similar. Also, figure 4.1 illustrates that there is no simple linear function that can map a P100 IPC value to a V100 IPC value. Though they are only one generation apart, the P100 and V100 have fundamental differences that can be seen when running a variety of applications between them. The V100 was developed with a core purpose of creating better performance for AI applications, though it’s not always the case, as shown here and among other benchmarks (10).

Nvidia’s NVprof is used to profile all applications, acquiring around 120 metrics, depending on the architecture, to give a detailed overview of an application’s performance on the GPU architecture. These are the metrics that are used as features to predict IPC on the given architecture.

4.2 Architecture Performance: IPC

Instruction per cycle (IPC) first came about in the development of CPU’s. The focus of predicting IPC was realized as IPC is a good indicator on whether the architecture is optimally performing. Additionally, because of the complexities and differences between CPU and GPUs, IPC is an easily available metric that can be traced across these two differing chip architectures.
<table>
<thead>
<tr>
<th></th>
<th>Kepler (K40)</th>
<th>Maxwell (M60)</th>
<th>Pascal (P100)</th>
<th>Volta (V100)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CPU</strong></td>
<td>IBM Power8 @2.2GHz</td>
<td>Intel Xeon E5-2670 @2.60 GHZ</td>
<td>IBM Power8 @2.2GHz</td>
<td>Intel Xeon E5-2699 @2.20GHz</td>
</tr>
<tr>
<td><strong>Computation</strong></td>
<td>3.5</td>
<td>5.2</td>
<td>6.0</td>
<td>7.0</td>
</tr>
<tr>
<td><strong>capability</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>SMs</strong></td>
<td>15</td>
<td>16</td>
<td>56</td>
<td>80</td>
</tr>
<tr>
<td><strong>Cores/SM</strong></td>
<td>192 SP cores/64 DP cores</td>
<td>128 cores</td>
<td>64 cores</td>
<td>64 SP cores/32 DP cores</td>
</tr>
<tr>
<td><strong>Texture</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Units/SM</strong></td>
<td>16</td>
<td>8</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td><strong>Register</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>File Size/SM</strong></td>
<td>256 KB</td>
<td>256 KB</td>
<td>256 KB</td>
<td>256 KB</td>
</tr>
<tr>
<td><strong>L1 Cache/SM</strong></td>
<td>Combined 64K L1+Shared</td>
<td>Combined 24KB</td>
<td>Combined 24 KB</td>
<td>128 KB Unified</td>
</tr>
<tr>
<td><strong>Texture Cache</strong></td>
<td>48KB</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Shared Memory/SM</strong></td>
<td>Combined 64K L1+Shared</td>
<td>96 KB</td>
<td>64 KB</td>
<td></td>
</tr>
<tr>
<td><strong>L2 Cache</strong></td>
<td>1536 KB</td>
<td>2048 KB</td>
<td>4096 KB</td>
<td>6144KB</td>
</tr>
<tr>
<td><strong>Constant Memory</strong></td>
<td>64 KB</td>
<td>64 KB</td>
<td>64 KB</td>
<td>64 KB</td>
</tr>
<tr>
<td><strong>Global Memory</strong></td>
<td>12 GB</td>
<td>8 GB</td>
<td>16 GB</td>
<td>16 GB</td>
</tr>
</tbody>
</table>

Table 4.1: Key specifications of selected GPUs of different generations
4.3 Active learning

In active learning, the overall model chooses the number of examples to learn the concept, often leading to a much lower amount of data points (11). Given a heuristic, such as finding the variance of predicted values compared to true values or another error measurement, the active learner can choose the most difficult points to predict. The there are other forms of creating the base set, the original training set batch are points randomly picked from the data set. The random forest model is then trained with this base set. The random forest model then predicts on the hold out set where the variance between the prediction and true value is calculated. The points are then ordered from highest to lowest based on their variance values. A batch of points with the highest variance are chosen as they are considered the poorly predicted points. These points are considered valuable as the model could not predict them well and will be added to the training set. The model is retrained with the
newly formed training set (base set and new points). After the learning cycle is completed, the points are once again ranked dependent on their variance score and a new batch of points are added to the initial training set. After each addition to the training set, the model is retrained and scores of the hold out set are recalculated until the specified training size and learning cycles are completed. The carefully curated set will have the optimal number of points that allow for accurate predictions.

4.4 Deephyper, NAS, and Balsam

The DeepHyper, NAS, and Balsam frameworks, all developed at Argonne National Laboratory, were vital in conducting a scaled search over millions of deep learning models (12). DeepHyper is a scalable framework designed to search the hyper-parameter space of deep neural-network models. DeepHyper also includes an integrated neural-architecture search (NAS) mechanism, which enables the automated generation and testing of neural-network models. The NAS feature is tightly coupled with Balsam, a workflow manager which utilizes a persistent job/task database to efficiently utilize leadership-scale distributed supercomputing resources. Balsam dynamically packages tasks into ensemble jobs and manages the end-to-end scheduling life cycle, ensuring fault tolerance along the way. Additionally, Balsam allows for a complex multi-workflow solution with little user configuration.

As validated using a class of representative cancer data (13), DeepHyper-NAS automates the generation of deep-learning models using a reinforcement-learning. To execute an NAS workflow, DeepHyper results are used to dispatch reward-estimation tasks, based on $R^2$, to Balsam. After the architecture search is completed, there are between 15,000 to 30,000 distinct models generated, trained and tested. From this large pool of models, the estimated reward values are used to select the top 50 DNN architectures. The top 50 DNN architectures are then submitted to a post-training sequence, during which each model is thoroughly trained on the full training data.
In this work, we utilize this Deephyper-based NAS workflow to predict performance metrics on Nvidia GPU architectures. The use of this model-generation pipeline allowed us to test more than one million neural architecture models.
CHAPTER 5

INTRA-ARCHITECTURE IPC PREDICTION

As seen in past work, it is possible to predict performance metrics within the same architecture using a variety of techniques and tools. Here we present a deep learning method that allows for IPC prediction of applications run on the P100 architecture, given the NVProf metrics to collect 46,039 data points, each containing 116 performance metrics. This sequential deep learning model has a normal kernel initialization with 1 input layer and 2 hidden layers with all layers using the ReLU activation function, except the output layer. The ReLU activation function is used as no values are to be less than zero and it showed better convergence results compared to using sigmoid activation functions. The deep learning model uses the adam optimizer and uses mean squared error for loss. The Adam algorithm calculates an exponential moving average of the gradient and the squared gradient, and the parameters beta1 and beta 2 control the decay rates of these moving averages (14). The model is trained for 100 epochs.
6.1 Cross architecture memory bound prediction

We first look at cross architecture memory bound prediction. In addition to the P100 data points collected for the intra-architecture prediction step, 32,291 V100 data points are collected using the NVProf profiling tool. Only V100 data points that had corresponding P100 data points were used at this point. We explore whether an application becomes memory bound from one architecture to another. Applications that require processing large amounts of data, such as multiplying large matrices, would likely be memory bound. By Nvidia architecture standards, the definition of an application becoming memory bound on their architecture is of an application having a dram utilization of over 70% on the architecture.

In graph 6.1, you can see the differences in dram utilization between the two architectures in both scaled and un-scaled versions. There is no simple linear function that would map all P100 dram values to V100 dram values.

**Figure 6.1:** Dram read and write utilization on both P100 and V100 GPUs
In graph 6.2, the results of memory bound applications on both P100 and V100 applications are plotted along their IPC value. Here, IPC results are more spread out with a high DRAM total throughput on the V100 compared to the P100. Using a random forest classifier, given performance metrics derived from the NVProf profiling tool, we were able to predict whether an application run on a P100 would become memory bound on a V100 given the dram read and write performance numbers on the P100. The random forest classifier is created using a grid search with 110 max features. The random forest classifier uses the performance metrics given by the NVProf profiler as input features and the results of dram utilization as a label to whether or not the application is memory bound. The classifier is trained on a set of P100 metrics with the applications corresponding V100 target values. It is then tested on the validation and test sets to confirm that the model is not over-fitted and
performs well on untested data.

6.2 Two stage Framework for IPC prediction

With the use of Argonne’s DeepHyper framework, we tested thousands of deep learning models. This framework was tested in combination with an active learning model obtaining a specified training set given to DeepHyper. The active learning model uses a random forest regressor to predict the IPC of a given application. The random forest is trained on 115 performance metrics, excluding the application’s IPC, to predict IPC of the given application run (intra-architecture IPC prediction). The active learner is run several times to acquire training sets of sizes ranging from 2.5\% to 20\% of the total testing data. The resulting training sets are used as the training sets in the DeepHyper framework. To see the benefits or lack of benefits from the addition of the active learner, equivalent sized training sets are created. These training sets are randomly chosen data points from the total data set.

The data distribution of the application points selected between random selection and active learning selection can be seen in figures 6.5 and 6.3. The percentage breakdown for each selection process is seen in figures 6.5 and 6.4. In particular, the random selection process gives a better overlook of the total application data. The active learning selection process, reduces the amount of points across the other applications, as the training set gets bigger. This reduction of points among the other applications can be understood by seeing that the backprop application is difficult to predict and therefore obtains the highest variance when predicted. Additionally, the quantity of backprop data dominates the data set sampling insuring a high amount of backprop application specific points will be ranked high.
Given the information from a singular DeepHyper neural architecture search and the created training sets from the active learner, we now scale the neural architecture to span the search for each training set. There are 24 training sets - 12 random and 12 active learning created sets. Deephyper’s NAS framework utilizes balsam’s framework to test and train these models in parallel. The neural architecture search produces between 15,000 - 30,000 models for each training set.
Figure 6.4: Application data distribution of data points that were chosen at random
Figure 6.5: Normalized application percentage breakdown of data points created using active learning
Figure 6.6: Application data distribution of data points created using active learning
CHAPTER 7
EXPERIMENTAL RESULTS

Here we present the results of intra-architecture IPC prediction, memory bound inter-
architecture prediction, and inter-architecture IPC prediction. We also present the two
architecture structures created by DeepHyper to illustrate the complexity compared to a
conventional model.

Figure 7.1: P100 IPC prediction using 451 training data points
Figure 7.2: P100 IPC prediction using 4,521 data points

Intra-Architecture IPC prediction (P100) - MAPE: 2.98, Training Points Used: 4,521

Figure 7.2: P100 IPC prediction using 4,521 data points

<table>
<thead>
<tr>
<th></th>
<th>Predicted: No</th>
<th>Predicted: Yes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actual: No</td>
<td>0.995</td>
<td>0.0048</td>
</tr>
<tr>
<td>Actual: Yes</td>
<td>0.0174</td>
<td>0.983</td>
</tr>
</tbody>
</table>

Table 7.1: Confusion matrix showing results of memory bound random forest classifier that predicts whether an application will be memory bound going from the P100 to V100 GPU architecture.
Figure 7.3: MAPE of each framework across applications tested

Figure 7.4: IPC prediction results of DeepHyper model using a training set with randomly chosen data points.
Figure 7.5: IPC prediction results of DeepHyper model using a active learning curated training set.
Figure 7.6: Prediction of backprop IPC using DeepHyper model with active learning chosen training set.
Figure 7.7: Prediction of Stream application using model returned Deephyper using a training set with randomly chose data points.
Figure 7.8: Prediction of Stream application using model returned Deephyper using a training set curated by the active learning model
Figure 7.9: Prediction of Stream application using model returned DeepHyper with random selection.
Figure 7.10: Conventional deep learning architecture layout.
Figure 7.11: DeepHyper + Active learning architecture layout.
8.1 Intra-architecture performance

We use the intra-architecture IPC prediction experiment as a stepping stone to understanding the capabilities of predicting specific metrics from other performance metrics. As shown in figures 7.1 and 7.2, using both a big and small training set, there is very little error when trying to predict IPC given other performance metrics, within the same architecture.

8.2 Memory bound cross-architecture prediction

As shown in the confusion table 7.1, the prediction results show both the false positive and false negative rates. This classifier can give a small insight on which applications will become memory bound that may not have been originally be memory bound. The accuracy of this memory bound classifier is .99. Our model shows that there are some applications, such as hybrid sort, that become memory bound going from the P100 to the V100. This is particularly useful find a developer to focus on different performance optimizations or possibly void the chip architecture all together. Knowing if an application will become compute bound spotlights areas of optimization. This knowledge also reduces time in porting the application onto untested architecture.

8.3 Inter-architecture IPC prediction

Overall the final DeepHyper model prediction does better than some models but still needs improvement. The bar graph 7.3, shows the mean absolute percentage error (MAPE), of each model type. The 'Old to New' bar is the MAPE of the P100 IPC compared to the V100 IPC without any changes to the P100 IPC. The 'Random Forest' is the MAPE of a random
forest regressor model that takes in the P100 metrics to predict V100 IPC for the same corresponding application. The random forest performance is on par with the deep learning model performance. The 'Random Forest + AL' bar is the MAPE for a random forest model that uses a training set created by an active learner discussed above. This particular model does not do well in applications such as kmeans and hybrid sort, but does one percentage point better than most of the other models for backprop. This could be due to the fact that the active learning selection has a concentrated focus on the backprop application points in comparison to the other applications.

The 'conventional DL' bar is the MAPE for a deep learning model, architecture shown in figure 7.10, developed by conventional methods. After many renditions, this model shows very little improvement over the other model types. The 'DL + Random' bar is the MAPE for a DeepHyper NAS created model using a training set created by randomly selected points from a data set. The 'DL + AL' bar is the MAPE for a DeepHyper NAS created model using a training set created by the active learning model. You can see the complexity of a DeepHyper model in figure 7.11. This is the architecture returned by DeepHyper using a training data set created by an active learner. Every model created by DeepHyper is as complex if not more complex than the one shown in figure 7.11.

When comparing these two DeepHyper created models, there are some cases where the random selection does significantly better than the active learning selection, kmeans by 15 percentage points and hybridsort by 48 percentage points. There are also cases where active learning cases does better than random selection, leukocyte by 13 percentage points and srad by 13 percentage points. Both do relatively the same in backprop, shown in graphs ?? and ??, even though the active learn selection model has a significant amount of backprop points in each training set, clearly seen in graphs 6.4 and 6.6. Both models did well with predicting IPC for stream, as seen in graphs 7.8 and 7.9. If we say a MAPE of under 5% is excellent, only one application prediction falls into that category across all models except
'Old to New'. These results show that active learning did not give beneficial improvements to the model and at times reduced the accuracy tremendously. These results also show that even after training and testing almost a million models, a more complicated heuristic is need to identify the relationship between these architectures.
CHAPTER 9
FUTURE WORK AND CONCLUSION

9.1 Future Work

Further work will examine several DeepHyper modifications such as allowing for a wider network in the neural architecture search optimization. The preliminary search had a small network search that could have potentially prevented obtaining better results. Additionally, incorporating the DeepHyper hyperparameter optimization framework has great potential at optimizing the current models for better performance. We would take the top performing DNN architectures and optimize the hyperparameters. DeepHyper has seen significant improvement in accuracy with this hyperparameter optimization.

The current work looks at predicting only IPC. Further work will look into predicting metrics beyond IPC, such as DRAM utilization and other highly looked after metrics.

When optimizing and picking the best neural architectures, the deep hyper search looks at the optimization metric. The current method uses R2 as its optimization metric, which results in MAPE values that aren’t very good. Further steps will be taken into modifying the optimization metric to use MAPE instead. By basing the optimization metric on MAPE, instead of R2, we could potentially decrease the error between the prediction and true values.

Further steps will also be taken at improving the active learning model to have a better overview of the entire data set versus only the points with the highest variance.

9.2 Conclusion

Developing a thorough understanding of new chip architectures is often a tedious and time consuming endeavor. In contrast, the development of a data-driven model can be much less time consuming in the case that relevant performance data is readily available. In this work, we have highlighted the current opportunities and limitations of a purely data-driven
approach. We have shown that classical machine learning (i.e. random forest) can be used to successfully predict if an application will become memory bound when switching between the P100 and V100 GPU architectures. That is, both deep-learning and random-forest classification was used to predict applications that will become memory bound on lightly tested architectures, with 99% accuracy, with no changes to the original source code.

Although data-driven modeling was found to be sufficient for the task of classifying if a kernel will become memory bound, the prediction of specific performance numbers proved much more challenging. For this reason, this work was largely focused on the optimization of deep neural-network models for the intra-architecture prediction of IPC. Like the memory-bounded classification task discussed above, deep-learning models were also found to accurately predict IPC when trained on performance metrics collected for the same GPU architecture. However, we also found that two distinct GPU architectures, though only one generation apart (and with many similar features), have a complex relationship that not even a search over one million neural-network models could capture. Although our automatically-generated deep-learning models outperformed other notable performance-prediction techniques, the overall accuracy remains insufficient for practical use.
REFERENCES


