Faculty of Engineering Sciences

Heidelberg University

Master Thesis
in Computer Engineering
submitted by
Constantin Nicolai
born in Bretten, Germany
Day/Month/Year Here

YOUR TITLE HERE

This Master thesis has been carried out by Constantin Nicolai
at the
Institute of Computer Engineering
under the supervision of
Holger Fröning

ABSTRACT

Briefly summarize the contents of your work in 150-250 words. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

ZUSAMMENFASSUNG

Deutsche Version. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nice quote here.

— Some Author

ACKNOWLEDGMENTS

Your acknowledgments here if desired

CONTENTS

1 INTRODUCTION

1.1 MOTIVATION

The global increase in usage of machine learning applications illustrates an acceleration in adoption across both industry and the private sector. The unfathomably large energy costs tied to this broader adoption have already prompted a change in public sentiment towards energy energy infrastructure. Plans for building trillion-dollar data centers are emerging, necessitating the re-commissioning of previously decommissioned nuclear power plants, which were originally phased out as part of nuclear energy reduction efforts. This reversal of nuclear phase-out policies underscores the significant infrastructural and political pressures exerted by the energy requirements of machine learning technologies.

In this landscape it is more pressing than ever to gain insight into the roots of the energy costs in order to optimize future developments on an informed basis.

In order to facilitate a more informed pairing of workload and GPU we introduce a framework to help guide the decision towards an optimal choice. This way regardless whether the fastest execution or the smallest energy footprint is desired, the informed choice enabled by our framework prevents wasteful computation.

1.2 PROBLEM STATEMENT

While a considerable amount of previous work has been done in profiling and prediction of neural network performance, no prior work covers the same cases and performance metrics targeted in this work. Therefore, our study investigates both training and inference cases covering both execution time and power consumption.

This contribution is valuable because in most cases where a new model architecture is designed or an existing architecture is adapted, both the training and the inference efficiency are relevant at some point of the models lifespan. At the same time, latency or power envelope requirements may be fluent between the training and inference stages, necessitating the study of both performance metrics.

1.3 SCOPE

This can only be written, when research is finished.

1.4 CONTRIBUTIONS OVERVIEW

1.4.1 Dataset Collection

In our first contribution we introduce our method for collecting profiling data and creating a coherent dataset by processing the collected data.
The profiling measurement is executed via a python script, utilizing the Pytorch Benchmark library[1] to conduct the execution time measurements. The power measurements are conducted using the command-line tool nvidia-smi[2] being run in the background while the benchmark is performed.
In order to enable repeatability the execution times and power readings, along with their respective measurement errors are stored together with the Pytorch objects for profiled operations, along with a sensible selection of related metrics.

1.4.2 Prediction Model

In our second contribution we present our prediction model. Based on the predictions for the operations occurring in a neural network we can infer a prediction for the complete network. We can therefore provide insight into which of the studied GPUs is most suitable for a given neural network, depending on the metric of interest. This way we can identify both the GPU which can execute the neural network in the fastest time, and the GPU which results in the smallest energy consumption and accompanying heat output.

1.4.3 Validation

In our third and final contribution we investigate the quality of our dataset of measurements for individual operations by comparing the sum of individual operations against a full neural network run for both execution time and energy consumption.
In the second part of this contribution we assess the accuracy of our predictions both for individual operations as well as for full neural networks. Both for the individual operations as well as for the full neural networks this is achieved by comparing the predictions to

[1] https://pytorch.org/tutorials/recipes/recipes/benchmark.html

[2] https://docs.nvidia.com/deploy/nvidia-smi/index.html

measurements acquired using the same measurement methodology used to collect the initial dataset.

2 STATE OF THE ART AND RELATED WORKS

2.1 STATE OF THE ART

The challenge of predicting neural network performance has invited a plurality of approaches. Apart from the methodological approaches they also differ in a number of aspects. While execution time is commonly the metric of choice, only few go further and also study metrics like power consumption and memory footprint. Another important distinction is the workload studied in the work, more specifically, whether both training and inference are studied. For practical reasons it is also relevant which machine learning framework is used and what hardware targets are required and can be predicted for. These many dimensions of possibility result in no work covering all possibilities, but allows for many different approaches which have use cases in a given situation.

2.2 RELATED WORK

Kaufmann et al. take an approach of performance modeling by means of the computation graph. They are however limited to the Google Tensor Processing Unit in this work.
Justus et al. take an approach exploiting the modular and repetitive nature of DNNs. Given the same operations are repeated over and over in training, often only varying in a few key paramters, these execution time for these base building blocks is measured. This is then done for one batch in the training process and generalized to the whole training process from there. There is however no presentation of the methodology for the execution time measurements.
Qi et al. present PALEO which employs an analytical approach towards predicting the execution for both training and inference of deep neural networks. The analytical approach brings both advantages and disadvantages with it. It does not require a dataset of measured execution times as a training set in the same way many other works do, but on the other hand it also is based on more fixed assumptions about the DNN execution than a more data driven approach.
Wang et al. approach with a mult-layer regression model to predict execution time for training and inference. Their work is however rather limited in terms of hardware targets and different DNNs studied.
Cai et al. focus their work, NeuralPower, on CNNs running on GPUs. For each target GPU, they collect a dataset and fit a sparse polynomial

regression model to predict power, runtime, and energy consumption. While NeuralPower achieves good results, its usefulness has become limited due to its exclusive focus on CNNs, as other DNN architectures have grown in popularity.
Gianitti et al. also exploit the modular nature of DNNs in their approach. They define a complexity metric for each layer type, optionally including backpropagation terms, allowing them to predict execution times for both training and inference. However, their method faces significant limitations, as the complexity metric is only defined for a specific set of operations, making it incompatible with networks that include layers not covered in the original work. As a result, their approach is essentially limited to classic CNN architectures.
Velasco-Montero et el. also take the familiar per-layer approach. Their predictions are based on linear regression models per type of layer, but again for a specific set of predefined operations. Given their focus on low-cost vision devices these restrictions are reasonable, but limit generalizability.
Sponner et al. take a broad approach in their work. It works in the TVM framework giving it high flexibility in target hardware and studied metrics. It is in fact the only work to include execution time, power consumption and memory allocation. Given the automated data colletion used to create the dataset basis for the predictions, there are also few limitations to the networks that can be studied with this. The predictions are based on an extremely randomized tree (ERT) approach with XGBoosting applied. The only major drawback for this work is its limitation to only study inference, due to TVMs limitation to inference.

2.3 RESEARCH GAP

With all these very different works no single one was able to cover all possible angles to interest, although Sponner et al. got rather close. But given the current landscape of available publications our work will focus on finding the best GPU for a PyTorch job. In order to achieve that, we will cover execution time, power and energy consumption and will provide inference and training predictions for these metrics. Our approach also employs a different method of automatic dataset collection, which allows for a broad field of study. In order to obtain power readings are collected directly through nvidia-smi. While due to the scope of this work this limits us to Nvidia GPUs, the methodology could just as well be applied to any other hardware target which supports reporting power readings.

3 DATASET COLLECTION

This contribution outlines the method used to collect a profiling dataset. The dataset serves as a training set for the prediction model. The parameters covered are various Torchvision models and input sizes, execution time and power, both the inference and the training case as well as different GPUs and various GPU clocks.

3.1 OPERATIONS

In order to increase generality our approach exploits the layerwise structure of DNNs. This is achieved by working on the layer level rather than on the model level. In order to be rigorous we need to be clear on the terminology. The workload and its characteristics depend on the layer and its input features, just like it would on the DNN and the input dimensions on the model level. We are interested in most general objects with an identical characteristic workload. On the layer level those will be layers with specific input feature dimensions and layer settings. We will refer to these objects as operations. On the model level those will be DNNs with specific input image dimensions, which we will refer to as model-input sets.

The units for which we will make predictions later are individual operations. To this end, we aim to collect a dataset of operations and profile the execution time and wattage for each one. To ensure that the dataset reflects operations encountered in real-world scenarios, we select a number of representative model-input sets from models of the Torchvision library. The set of operations which we will be profling, is the set of all operations occuring in any of these model-input sets.

Extracting this set is automated via scripts, but can just as well be done by hand if so desired. With this set at hand we can dive into the actual profiling.

3.2 TIME PROFILING

The time profiling is performed using the benchmark utility from `torch.utils.benchmark` called Timer. This enables us to run our benchmark function with the specific layer and input features as input parameters, allowing us to run it for each operation.

To achieve more stable results and inprove the simultaneous power profiling it is desirable to run the benchmark for each operations

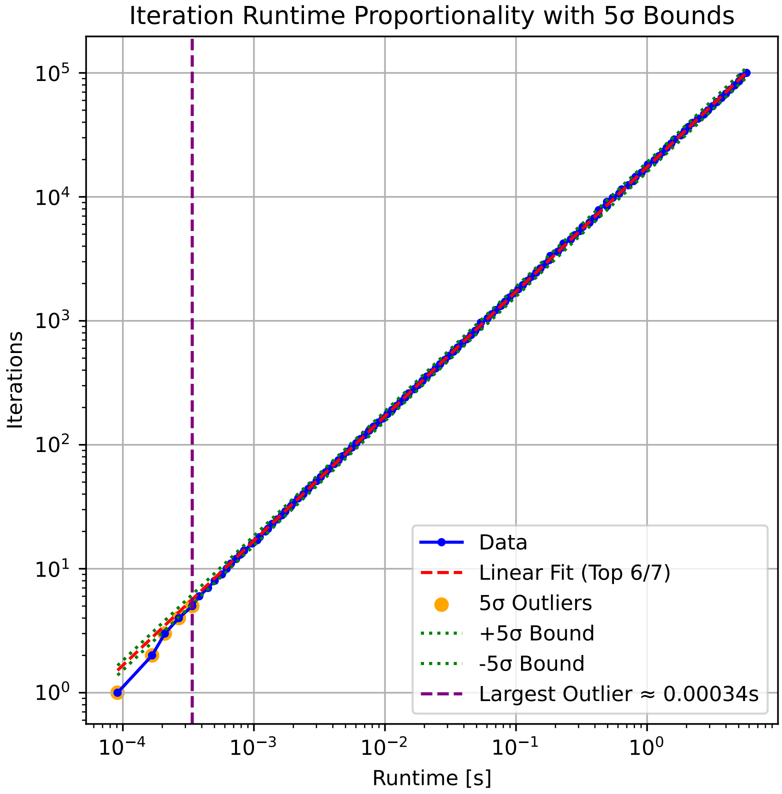

Figure 3.1: This plot shows the proportionality of the runtime with the number of iterations ran for an operation. A linear fit is performed to the $\frac{6}{7}$ of data points with most iterations. From this fit the standard deviation for the linear section is found. This allows us to identify the largest outlier below which the proportionality breaks. In order to do this we search for the largest runtime in the 5σ outliers.

for at least a few seconds. This is achieved by setting a minimum runtime for the benchmark to aggregate statistics. Different workload characteristics and different computational capabilities depending on the GPU configuration being profiled result in different requirements for individual profiling runs in order to ensure sufficient repitions in the measurement process. In order to accomodate this requirement, the runtime of the measurement is configurable via a command-line parameter.

For the initialization of the operations we have to be careful to avoid two extreme cases in order to achieve a sensible approximation of the execution characteristics within a neural network. The first case we want to avoid is initializing one instance of the operation and performing each benchmark loop on that instance. This would result in hitting the cash every time and would not be representative. The second extreme we want to avoid is initializing a new instance for each loop of the benchmark run, which adds the initializition overhead and the larger latency of having to access the GPU main memory every time, skewing our results in the opposite direction. We have therefore chosen the middleground approach of initializing a sizable block of operation instances with random weights and biases, which is then looped over by the benchmark kernel, resulting in some reuse of layers, stressing the memory system in a sensible manner.

3.2.1 Inference

The central difference between inference and training profiling appears in the design of the benchmark function.

For inference profiling we put our operations into `eval` mode and call the operations within a `torch.inference_mode` environment, which is equivalent to the exection in a typical inference forward pass through a model. Within the loop we call the operator on a preinitialized random input tensor for the forward pass. The input tensor is a single instance, as it represents the activations from the previous layer which can be expected to live in high level memory in our memory hierarchy. Since this forward pass is everything we want to profile for inference, this benchmark loop is sufficient.

3.2.2 Training

For the training profiling we put our operations into `train` mode and omit the environment used in the inference case. In order to portray the training process for a single operation, we need to run a forward pass and a backward pass. The forward pass is handled identically. In order to execute the backward pass we need a substitute for the gradients flowing backward trough the model. This is achieved by preinitializing a random torch vector with the dimensions of our operation's output which we can then call the backward pass on. Other than that, we only have to make sure the runtime keeps the gradients for all operation instances and keep resetting the input vector gradients for each loop iteration.

3.2.3 Proportionality

Since we are assuming the number of benchmark iterations and the resulting total runtime to be proportional, we need to test that assumption. In order to do that we plot their relationship in Figure 3.1. By identifying how short the total runtime needs to become for the linear relationship to break, we can find a lower bound to our desired benchmark duration for each operation we study. The result of this investigation is that as long as we stay above 100 ms we are on the safe side. In our actual measurements every operation is benchmarked for at least several seconds, even though the exact duration may vary depending on the computational intensity of the specific operation.

3.3 ENERGY PROFILING

The energy profiling is conducted by measuring the power in watts. Combined with the time measurements this gives us the energy results.

We are starting `nvidia-smi` as a background process and logging the power readings into a temporary csv file, which is evaluated later in the script to extract the statistics relevant to our purpose. On a higher level abstraction, a second instance of `nvidia-smi` runs during the complete benchmark process. This log can be compared to the timestamps included in the dataset of profiled operations, in order to investigate surprising annomalies in the results. However, the results relevent to our power profiling are calculated instanly from operation specific temporary log file. In terms of direct measurement there is no additional complexity involved for the power profiling. Still, some preparation and processing after the fact is necessary to ensure robust results.

As can be seen in Figure ?, showing the power measurement over time for alternating idle times and benchmark calls, the transition is not instant. There are some power readings in between the steady states of idle and benchmark.

In order to illustrate the existance of a startup effect without idle times between the benchmark runs, Figure ? shows five looped benchmark runs of the same benchmark overlayed. For each run we can observe the startup effect.

We do not want to keep this startup effect in our result, because it is a

result of our benchmark exection and not representative of the much more integrated execution pipeline in a neural network.
In order to keep it from affecting our results, we use a 3σ channel around the initial mean of the power readings and drop everything else.
As a second precaution, the script also employs some warmup runs in order to further mimimize the impact of the startup effect.

3.4 LOG FILE EVALUATION

The following section explains how we arrive at our results using the log files collected in the benchmark run. Below you can see a snippet from one of the log files. The fourth entry in each row is the power.

```
2024/10/10 13:18:58.369, 81, 145.99, 4396, 11264
2024/10/10 13:18:58.407, 81, 182.11, 4396, 11264
2024/10/10 13:18:58.428, 81, 182.11, 4396, 11264
2024/10/10 13:18:58.439, 81, 182.11, 4396, 11264
2024/10/10 13:18:58.490, 81, 178.50, 4396, 11264
2024/10/10 13:18:58.514, 81, 178.50, 4396, 11264
2024/10/10 13:18:58.538, 81, 178.50, 4396, 11264
```

We begin with the power evaluation. The log file is read in via `pandas`[1]. Any existing rows containing a non-numerical value are dropped from the dataframe. We then find the standard deviation for the power and drop all rows containing a power reading outside a $3\,\sigma$ range. The following formulae are used:

$$\overline{W} = \frac{1}{n}\sum_{i=1}^{n} W_i \tag{3.1}$$

$$\sigma_W = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(W_i - \overline{W})^2} \tag{3.2}$$

$$W_{filtered} = W \text{ such that } |W - \overline{W}| < 3\sigma_W \tag{3.3}$$

With n being the number of timestamps and W being the power.
The same two formulae are used to find the mean power $\overline{W}_{filtered}$ and standard deviation $\sigma_{\overline{W}_{filtered}}$ of the filtered power.
Both the total runtime t_{tot} and its standard deviation $\sigma_{t_{tot}}$ are provided by `torch.utils.benchmark`. With power and time evaluated, we find the total run energy E_{tot} and its error $\sigma_{E_{tot}}$.

[1] pandas

$$E_{tot} = \overline{W}_{filtered} \cdot t_{tot} \tag{3.4}$$

$$\sigma_{E_{tot}} = \sigma_{\overline{W}_{filtered}} \cdot t_{tot} \tag{3.5}$$

From this, we find the time per iteration t and the energy per iteration e, as well as the error for the time per iteration σ_t and for the energy per iteration σ_e with the number of iterations being N.

$$t = \frac{t_{tot}}{N} \tag{3.6}$$

$$e = \frac{E_{tot}}{N} \tag{3.7}$$

$$\sigma_t = \frac{\sigma_{t_{tot}}}{N} \tag{3.8}$$

$$\sigma_e = \frac{\sigma_{E_{tot}}}{N} \tag{3.9}$$

3.5 GPU CLOCKS

4 SECOND CONTRIBUTION

The second contribution goes here.

5 DISCUSSION AND OUTLOOK

Summary and discussion of the results and outlook/future work.

a | APPENDIX

Appendix here

ERKLÄRUNG

Ich versichere, dass ich diese Arbeit selbständig verfasst habe und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Heidelberg, den Day/Month/Year Here

Constantin Nicolai