Profiling Deep Networks in Resource-constrained Systems

Hans Brouwer
under supervision of Dr. Lydia Chen, Masoud Ghiassi & Bart Cox

Introduction

Execution time and memory use are crucial to understand which parts of a program need optimization. However, a major problem with profiling these metrics is that the instrumentation itself has performance impacts. This effect is especially strong on memory- and compute-constrained devices.

In this research project, the EdgeCaffe framework (a pipeline of multiple deep neural networks targeted at edge devices) was instrumented to determine the memory requirements of layers in each network as well as the scaling of execution time with different memory limits. The goal was to build a model to predict the factor by which execution time increases versus an unconstrained environment based on a description of the networks to be executed and the memory limits.

The baseline that EdgeCaffe seeks to build upon stems from the observation that execution of fully-connected layers is constrained by loading their parameters into memory and that the execution of convolutional layers is constrained by the CPU capacity to perform the necessary matrix multiplications. Mathur et al. presented the strategy in their paper, DeepEye [3], that schedules these two classes of layers onto two separate execution threads that can work in parallel.

Methodology

To simulate differing memory restrictions, the EdgeCaffe pipeline was run in a Linux kernel Control Group (cgroup) [4]. A limit on the allowed memory can be set for each cgroup which forces the program to use swap space when exceeding it. The Linux kernel keeps track of various memory & execution time metrics for each cgroup, which can be polled from a separate process. This method is asynchronous, however, and cannot achieve a sampling rate high enough to accurately measure high frequency allocation patterns.

To refine the measurements synchronously, the pipeline was instrumented with Google PerfTools [1] (gperftools). Gperftools links the binary with a memory allocation library, tcmalloc [2], which the gperftools heap profiler dumps information from at set intervals (e.g. every 100 MB allocated). These heap dumps can then be analyzed after the fact and linked to the layers of each network that allocated the memory. However, this method does not take into account overhead of the kernel process itself.

Conclusion

Using both cgroup & gperftools memory information together and comparing their results with GNU Time, which gives the kernel's maximum physical memory usage, a complete picture of per layer memory requirements can be pieced together.

The work to build a model is still ongoing, but some obvious trends are visible in the data. These can hopefully be leveraged to build a complete picture of neural network performance under varying memory constraints.

Results

While the results are still preliminary, the experiments thus far seem to support DeepEye's observation. The data also show that not only is execution memory-bandwidth-limited, but this problem is also exacerbated with stricter memory limits.

The figures below show this trend. On the left is the mean loading time over ten runs of each layer by parameter file size under different memory constraints. The obvious trend of loading time increasing with the size of the parameters to load is immediately apparent. More interestingly, the loading times increase with lower memory limits. This suggests that just loading the weights of the network is already incurring swapping, especially for the largest layers. On the right is the same plot but with execution time of the layers instead loading time. Here the trend is absent except for in outliers and some of the largest layers. This implies that the execution time of the majority of layers is unaffected by less memory being available.

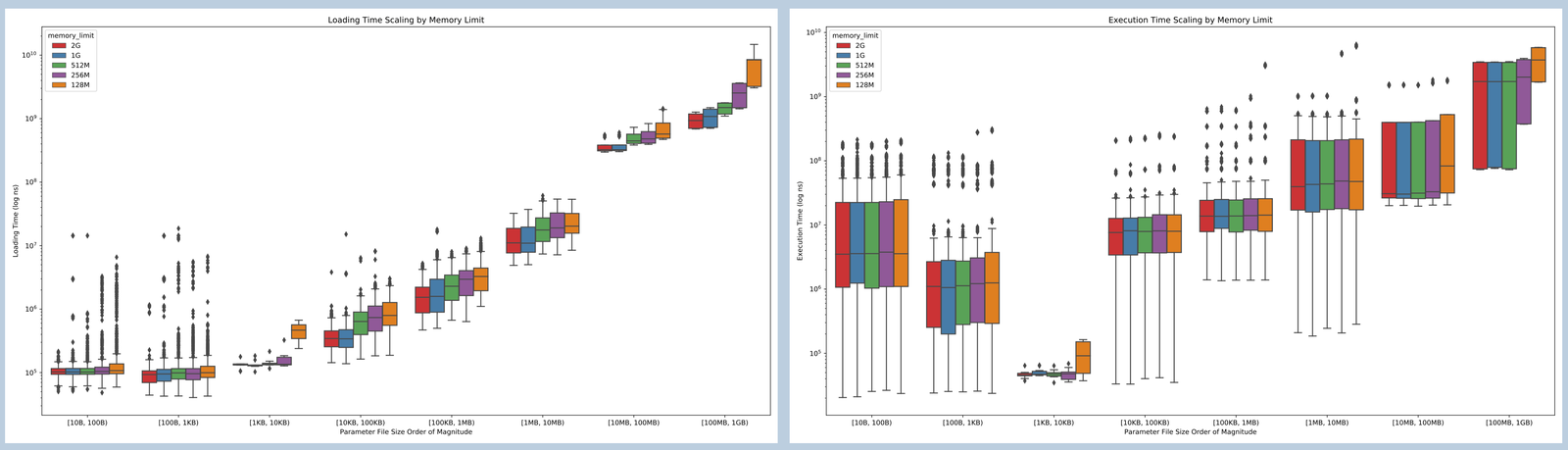

Figure 1: Loading time versus parameter file size under different memory limits (left); Execution time versus parameter file size under different memory limits (right)

On the memory profiling front, work is still ongoing to get a final number for each specific layer. The main issue is reconciling the resident set size (physical memory in RAM reserved by the process) measurements from the cgroup with the allocated memory measurements from the gperftools heap dumps.

Below is a graph comparing the memory usage per layer as reported by a couple different measurement strategies. The blue lines are measurements before and after each loading and execution task of a layer. These do not show the peaks in usage seen in the green lines as they do not measure extra memory needed during execution (e.g. for intermediate results of convolution operations). The high-water mark from GNU Time is also plotted and shows the current discrepancy that exists even between the strategies that do measure this intermediate usage.

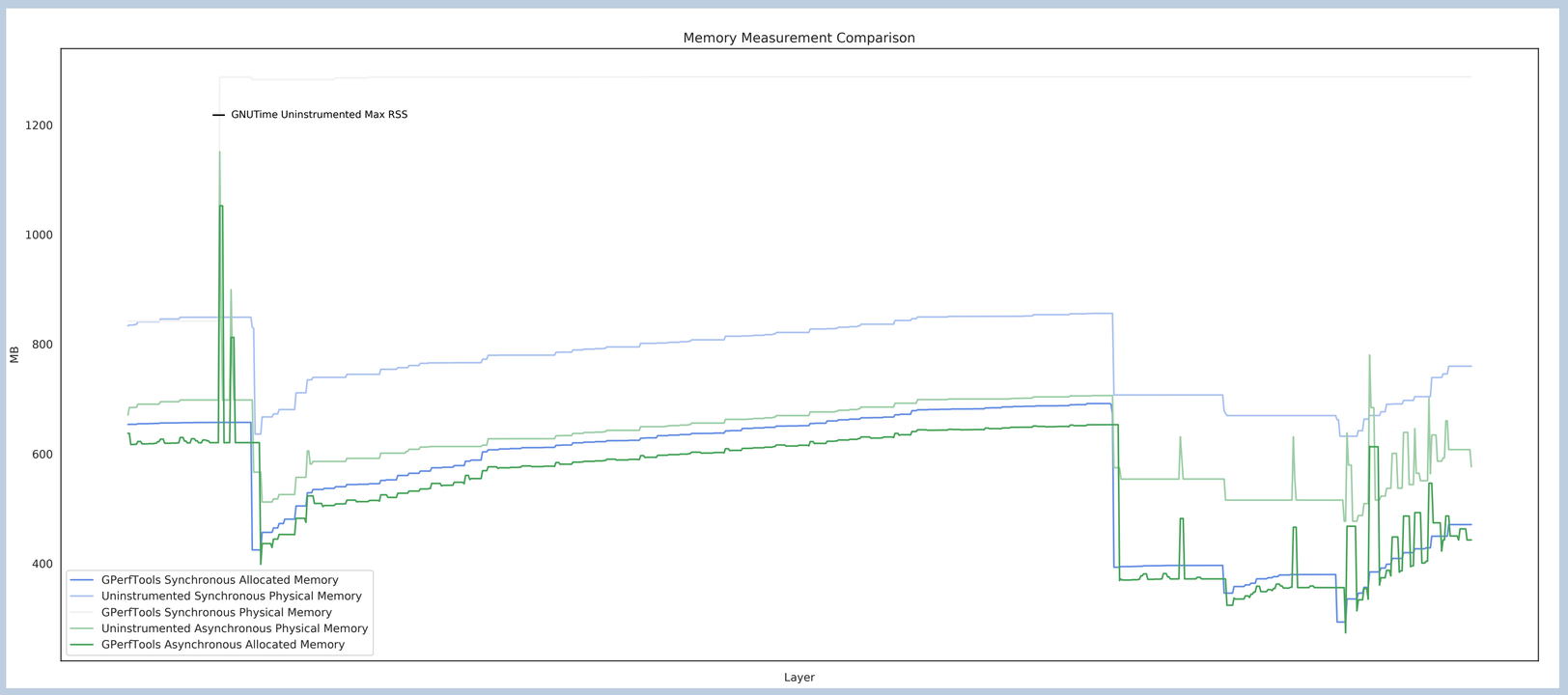

Figure 2: Comparison of measured memory usage during execution

References

[1] *Google PerfTools*. URL: https://github.com/gperftools/gperftools.

[2] *Google Tcmalloc*. Google, Apr. 29, 2020. URL: https://github.com/google/tcmalloc (visited on 04/30/2020).

[3] Akhil Mathur et al. "DeepEye: Resource Efficient Local Execution of Multiple Deep Vision Models Using Wearable Commodity Hardware". In: *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services*. MobiSys '17. Niagara Falls, New York, USA: Association for Computing Machinery, June 16, 2017, pp. 68–81. ISBN: 978-1-4503-4928-4. DOI: 10.1145/3081333.3081359. URL: https://doi.org/10.1145/3081333.3081359 (visited on 04/20/2020).

[4] Paul Menage. *CGroups*. URL: https://www.kernel.org/doc/Documentation/cgroup-v1/cgroups.txt.