Performance Characterization of State-Of-The-Art Deep Learning Workloads on an IBM "Minsky" Platform

Mauricio Guignard
FAMAF, Universidad Nacional de Córdoba
mguignard311@famaf.unc.edu.ar

Marcelo Schild
FAMAF, Universidad Nacional de Córdoba
marceloschild90@gmail.com

Carlos S. Bederián
FAMAF, Universidad Nacional de Córdoba
CONICET
bc@famaf.unc.edu.ar

Nicolás Wolovick
FAMAF, Universidad Nacional de Córdoba
nicolasw@famaf.unc.edu.ar

Augusto J. Vega
IBM T. J. Watson Research Center
ajvega@us.ibm.com

Abstract

Deep learning algorithms are known to demand significant computing horsepower, in particular when it comes to training these models. The capability of developing new algorithms and improving the existing ones is in part determined by the speed at which these models can be trained and tested. One alternative to attain significant performance gains is through hardware acceleration. However, deep learning has evolved into a large variety of models, including but not limited to fully-connected, convolutional, recurrent and memory networks. Therefore, it appears difficult that a single solution can provide effective acceleration for this entire deep learning ecosystem.

This work presents detailed characterization results of a set of archetypal state-of-the-art deep learning workloads on a last-generation IBM POWER8 system with NVIDIA Tesla P100 GPUs and NVLink interconnects. The goal is to identify the performance bottlenecks (i.e. the accelerable portions) to provide a thorough study that can guide the design of prospective acceleration platforms in a more effective manner. In addition, we analyze the role of the GPU (as one particular type of acceleration engine) and its effectiveness as a function of the size of the problem.

This research was developed, in part, with funding from the Defense Advanced Research Projects Agency (DARPA). The views, opinions and/or findings expressed are those of the authors and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

1. Introduction

The current success of deep learning techniques for machine learning is directly related to three complementary trends: the progress in new algorithms, the availability of big amounts of labeled data and the increasing computational power. Improving one of these areas usually demands improvements in the others. In particular, it has been noticed that research productivity is inversely proportional to the turnaround time of a deep learning experiment. While a few days is considered as tolerable, weeks are considered as "progress stalls" and experiments that take about a month are simply not worth running [1].

The huge computational demand from existing deep learning methods is driving a variety of new hardware solutions that emerge as deep learning application platforms. In recent months, platforms like Google's tensor processing unit (TPU) [2], NVIDIA's DGX-1 [3] and IBM's "Minsky" [4] have been announced or released, to mention just a few relevant examples. Hardware design has started to be shaped according to the needs of deep learning models with performance improvements that range from 10 to 100 times over conventional computing systems. As a result, previously intractable research problems turned into overnight jobs, opening up new types of learning algorithms and research opportunities.

However, significantly higher levels of performance and power efficiency are necessary in computationally constrained environments, like mobile applications and the Internet of Things (IoT) — unmanned aerial vehicles (*drones*), driverless cars, and "wearable" devices, among others. These aggressive performance and efficiency targets can be provided through hardware acceleration and low-voltage operation. Authors of this paper are pursuing related efforts as part of DARPA's Power Efficiency Revolution For Embedded Computing Technologies (PERFECT) program [5], where one of the goals is to develop ultra-efficient (low-voltage) hardware accelerators for deep neural networks. With the objective of choosing the most representative deep learning models to guide the design decisions in PERFECT, we conducted a search to find a set of workloads which had a significant impact in deep learning research and that properly reflects the current state-of-the-art in deep learning algorithms, while still keeping the size of the set within a manageable number. This led our attention to Fathom [6], a set of workloads put together by a PERFECT-player group at Harvard University.

In order to contribute to the current state-of-the-art in hardware platforms for deep learning methods, we execute Fathom on an IBM "Minsky" platform to characterize its performance and identify potential bottlenecks. The Minsky platform used in this work consists of two POWER8 CPUs with ten cores each, and four NVIDIA Tesla P100 GPUs, interconnected via NVIDIA NVLink [7]. We conduct a high-level analysis to highlight execution characteristics in the context of the Fathom benchmark suite. We identify the types of operations that represent a significant amount of the total execution time (most of them related to GPU kernels), measure similarity between deep learning models, compare performance differences between training and inference, and understand the effects of parallel scalability. Specifically, this paper makes the following contributions:

- We provide a quantitative analysis of the computational behavior of the Fathom workloads when they are executed in CPU-only and CPU+GPU modes. This study can provide insightful guidelines for the design of effective deep learning accelerators within the DARPA-sponsored PERFECT project.
- We confirm the performance advantage exhibited by an IBM "Minsky" system when executing deep learning networks.

The remainder of the paper is organized as follows. Section 2 presents the Fathom benchmark suite and describes its eight constituent applications. Section 3 presents the POWER8 "Minsky" system used in this work. Section 4 describes the characterization methodology and the porting of the Fathom benchmark suite, and presents the performance characterization results. Finally, Section 5 presents our conclusions.

2. Fathom

Fathom is a set of eight reference implementations of state-of-the-art deep learning models. It is implemented using TensorFlow [8], an open source software library for numerical computation using data flow graphs. It allows computation deployment to one or more CPUs or GPUs. Each of the Fathom workloads are described below.

2.1. AlexNet

AlexNet is a deep neural network (DNN) used for image classification. It consists of 5 convolutional layers, some of which are followed by max-pooling layers, and three fully-connected layers with a 1000-way SoftMax function at the end. Its main contribution was showing the computational power of GPUs when training neural networks with non-saturating neurons, as well as the introduction of a "dropout" regulation mechanism. This mechanism consists of setting the output of hidden neurons to zero with a probability of 0.5. Those neurons do not contribute to forwarding and neither participate in back-propagation, which was an effective way of reducing over fitting [9].

2.2. Variational Autoencoder

Autoenc is a stochastic variational inference and learning algorithm that scales to large datasets. It is a flexible unsupervised model that makes statistical assumptions about compact representations of realistic inputs, in order to reconstruct such inputs, providing a way to both analyze and synthesize data. These models require training and stochastic sampling for proper inference operation [10], and are often used for dimensionality reduction, feature extraction, or generating data [11].

2.3. Deep Reinforcement Learning

DeepQ [13] is a deep reinforcement learning system that learns playing Atari games just by "looking" at pixels and scores. Its actions improve as it receives in-game feedback, which distinguishes it from regular supervised algorithms that just try to reproduce the perfect play. The model consists of a convolutional neural network that selects actions using 2-3 convolutional layers and 2-3 dense layers. It is trained with a variant of Q-learning, whose input consists of raw pixels and that delivers a value that estimates future rewards.

2.4. End-to-End Memory Networks

Memnet is a neural network with a recurrent attention model over an indirectly addressed external memory. The idea is to decouple the net's state from its structure. It behaves similarly to any memory network [14], but unlike these, Memnet is trained end-to-end, which drastically reduces the required supervision during training, making it available for more general

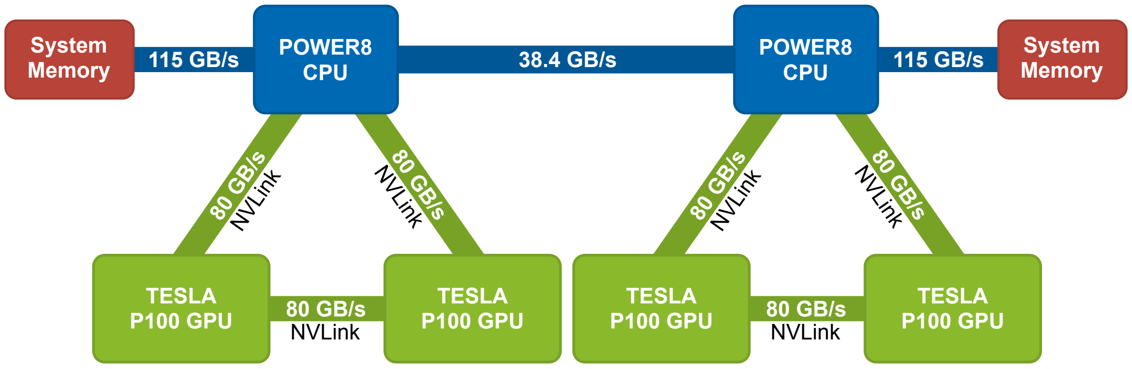

Figure 1: Minsky data flow diagram [4][12].

applications. It can also be seen as a recurrent neural network (RNN), since multiple computational steps are performed for each output symbol. The flexibility of this model allows its application to diverse tasks such as question answering and language modeling. It presents good performance in both cases, showing that the key concept of multiple computational hops produces better results [15].

2.5. Sequence-to-Sequence Translation

Seq2seq is an RNN used to perform translations. It uses a multi-layered long short-term memory (LSTM) pipeline to map an input sequence to a vector of a fixed dimensionality, and then another deep LSTM to decode the target sequence from the vector. It basically extracts the meaning out of the sentence and then builds the idea in another language. The core neural network is comprised of three 7-neuron layers through which word tokens flow unidirectionally. Sequence-to-sequence translation achieved the best-of-breed accuracy. The key feature that made it outperform other RNNs consisted of reversing the words in the source sentences [16].

2.6. Residual Networks

Residual networks are used for visual recognition tasks. They play a key role to enable deeper neural networks by directly attacking the problem where increasing the depth of the network degrades both training and validation errors. This is done by adding additional identity connections on each pair of convolutional layers, training them on the difference between their input and output [17].

2.7. VGG-19

VGG is a 19-layer convolutional neural network used for large-scale image recognition. Inspired by AlexNet, its main contribution is a thorough evaluation of convolutional networks of increasing depth but with very small convolutional filters. This has shown a significant improvement related to older techniques when increasing the depth to 16-19 layers [18], meaning that smaller convolutional filters are easier to train.

2.8. Deep Speech

Deep Speech [19] is a scalable speech recognition model. It consists in a RNN that uses spectrograms as inputs and learns to convert them into phonemes, which are distinct units of sound in a specified language that distinguish one word from another. This model implementation is very efficient and it was designed to run on GPU platforms.

3. IBM “Minsky” Platform

The Minsky platform is the result of a co-development effort between IBM and NVIDIA that pairs the strengths of the POWER8 CPU with four NVIDIA Tesla P100 GPUs [4]. These best-in-class processors are tightly bound with NVIDIA NVLink interconnects, which provide high bandwidth and low latency connections between CPUs and GPUs. NVLink delivers more than 2.5 times greater bandwidth than PCIe 3.0 16x and also allows the NVIDIA Tesla P100 GPUs to access the dual POWER8 CPUs massive memory bandwidth [7]. This combination makes Minsky capable of delivering extreme performance on traditional high-performance computing (HPC) applications as well as on high-performance data analytics applications. Figure 1 presents a diagram that shows how Minsky provides uncompromised data movement between GPUs and from GPUs to the main system memory without a bottleneck [7].

IBM’s POWER8 processor is a reduced instruction set computer (RISC) microprocessor with 10 cores with a maximum of eight threads each, fabricated using the 22-nm silicon-on-insulator (SOI) technology with 15 layers of metal. It has been designed to provide high single-thread performance and single-core throughput, achieving 1.5 times the single-thread performance of its predecessor (IBM POWER7) and twice its single-core throughput in many commercial applications [20]. In order to satisfy the high bandwidth requirement of nowadays HPC workloads, each processor has four memory channels running at 9.6 GB/s, capable of containing two *load byte* operations and one *store byte* operation in the load/store pipeline at a given cycle. This results in a 115.2GB/s memory bandwidth (4 channels x 9.6GB/s x 3 bytes = 115.2GB/s) between each CPU and its corresponding RAM memory. Each processor is connected to 256 GBs of RAM memory, making a total of 512 GB of RAM. However, the available bandwidth in the CPU-CPU communication is just 38.4GB/s, since it is implemented with an SMP X-bus (1 X-bus x 8 bytes x 4.8GHz = 38.4GB/s) [12], which makes it a possible bottleneck when trying to access memory from another processor.

As for the NVIDIA Tesla P100 GPU, it has been designed for challenging HPC and deep learning applications. Built with NVIDIA's Pascal architecture and with 3584 CUDA cores, it provides high floating point performance, delivering 21 teraflops (TFLOPs) of half-precision, 10.6 TFLOPs of single-precision and 5.3 TFLOPs of double-precision performance. This GPU has 16 GB of HBM2 stacked memory with an on-GPU memory bandwidth of 720 GiB/s [21]. Through the use of NVLink, this technology substantially accelerates time-to-solution for strong-scale applications.

4. Performance Characterization of Deep Learning Workloads on Minsky

As part of the characterization conducted in this work, we consider imperative to identify and understand the potential performance bottlenecks when Fathom is run on Minsky. However, the complexity, as well as the dynamically-compiled and the dataflow-oriented nature of the TensorFlow runtime system, limit the capabilities of most performance analysis tools. Consequently, we perform a high-level analysis of the different sections of the models defined as the distinctive functions in charge of the setup, training (policy gradient optimizer creation and application) and inference (operation execution and evaluation). In order to understand the total execution as a whole we also inspect the output of the NVIDIA *nvprof* profiling tool in the interest of describing the GPU usage characteristics of the various workloads by looking at the executed kernels, their *wall* times and the application programming interface (API) calls.

4.1. Porting Fathom to Minsky

The Fathom benchmark suite was originally published and released in September 2016 [6] and the TensorFlow API has rapidly changed since then. In order to take advantage of the available IBM PowerAI suite and its TensorFlow v1.0.1 framework optimized for the Minsky platform, we first upgraded Fathom to make use of TensorFlow's 1.0 API, considering that it was changed in ways that were not fully backward compatible with the deprecated TensorFlow 0.8.0rc0 API required by Fathom. For this reason, we refactored some package imports, function names and parameters while trying to stay true to the officially published release: Fathom 0.9-soft. As result of this work, we submitted a pull request to the Fathom's official GitHub repository, commits that were accepted by merging them to the upstream master branch on May 15 2017 [22] These changes allow Fathom to operate using the TensorFlow 1.0.x framework.

It is also important to mention that the version of Fathom used at the time of this work does not support multi-GPU execution. In order to keep our experiments consistent with the single-GPU results presented in Fathom's original paper [6], we disabled three out of four NVIDIA Tesla P100 GPUs in our Minsky system. This eliminated the overhead caused by TensorFlow while initializing all four GPUs, which despite being very small, represented about 14% of the total execution time of the smallest workload.

4.2. Defining the measurement procedure

This section describes the procedure followed to conduct the measurements presented in this paper. In order to understand the execution time breakdown, we partition each Fathom workload in different sections taking into account the standard model interface exposed by the Fathom applications. Although we identify several sections throughout our measurements, we focus just on those defined by the *init*, *setup* and *run* methods of the exposed interface since we are particularly interested in the behavior of the workloads when running in CPU-only *versus* CPU+GPU modes.

We use Python's *time.time()* function to measure the execution time of the different sections. In the context of the Linux operating system, *time.time()* provides enough resolution for the sake of our analysis, given that it has a much more precise granularity than 1/100th of a second [23]. Similarly, we build a simple C program that runs all the workloads and measures the elapsed time of each execution using the *omp_get_wtime()* function provided by the OpenMP library, confirming first that it has an 1e-9 second granularity using the *omp_get_wtick()* function of the same library [24]. This allows us to remove the one-time overhead related to the initialization of Python, TensorFlow and the GPUs.

As mentioned in the previous subsection, since none of the workloads use more than one GPU, we disable the other three GPUs in all the CPU+GPU measurements by setting the CUDA_VISIBLE_DEVICES environment variable to "0". In this way, we ensure that the measurements do not include avoidable GPU overhead that could otherwise compromise the results.

Finally, since we perform our experiments in a

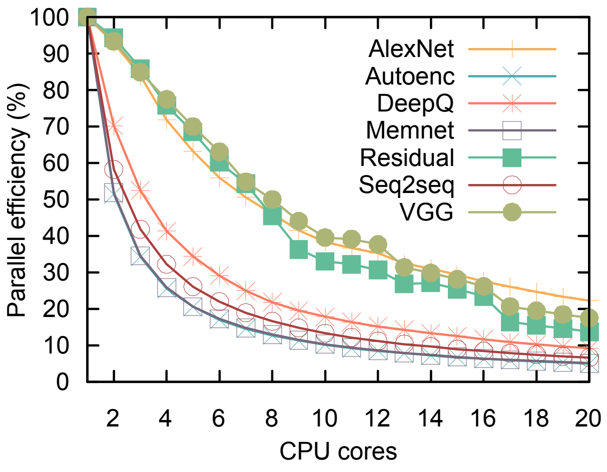

Figure 2: Parallel efficiency of the Fathom models.

cluster that uses the IBM Spectrum LSF workload manager, we execute 10 warm-up runs and 200 measurement runs for each workload with the goal of keeping system noise to a minimum.

4.3. Parallelism of the workloads

Since many of the leading operations in deep learning workloads are susceptible to parallelization, and nearly all of the existing work on architectural support for these models involves parallel hardware to some extent, we measure the CPU parallel scalability of the models to understand more deeply the benefits of using multi-core systems.

Figure 2 presents the parallel efficiency of the workloads measured as the speedup divided by the number of CPU cores used. We use the default TensorFlow thread pool configuration for the underlying Eigen library and increase the CPU affinity setting to restrict the number of available cores from 1 to 20. As GPU-accelerated computing allow us to offload compute-intensive portions of the applications to the GPUs, we find indispensable to understand how the workloads behave as the CPU core count increases, taking into account that the massively parallel architecture of the GPUs consists of thousands of small but more efficient cores designed for handling multiple tasks simultaneously.

As a result of this analysis, we can see that AlexNet, Residual and VGG exhibit relatively good scalability. However, as parallel resources are applied to the network, the parallel efficiency diminishes in relative importance in accordance with Amdahl's law [25]. On the other hand, Autoenc and Memnet do not parallelize well since the main operations, albeit frequent, operate on small, "skinny" tensors which trip-count is not large enough for thread-level parallelism. Memnet's behavior, in particular, is not unexpected since it is somewhat of an odd network to measure in the first place, considering that it was conceived with a research objective in contrast to, let say, VGG which has had much more effort put into fast execution. This in conjunction with the fact that the operations that Memnet carries out are not so much optimized in TensorFlow, since it is more a mixture of arithmetic, are the reason why we observe such a shallow parallel efficiency and performance in our measurements.

4.4. Measurement and analysis

First, it is necessary to mention that since we are not able to obtain the TIMIT corpus [26] due to licensing-related restrictions, in order not to detach from Fathom's original paper by replacing the dataset used in its performance breakdown, we exclude Deep Speech from the measurements and analysis presented below.

Figure 3 shows the execution time of the Fathom workloads normalized to the CPU-only mode. Figure 4 presents the execution time variance of the workloads normalized to the average CPU-only execution time for each of them. At first sight it is visible that all the Fathom workloads that depend on the ImageNet dataset [27], that is to say AlexNet, VGG and Residual, have extensive performance benefits when running on GPU. This behavior is in line with the total time spent by each workload, being the performance speedups more meaningful in those models where computational demand is higher.

This partially explains the deep learning community tendency of using GPUs to improve the object

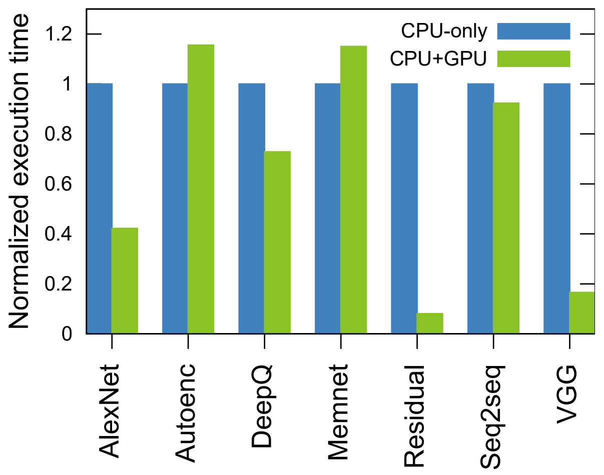

Figure 3: Performance of the Fathom workloads relative to the CPU-only total execution time. CPU+GPU executions use only one GPU. Relative performance differences indicate the benefits and disadvantages exhibited by each workload when running in CPU-only or CPU+GPU modes, using the default Fathom settings.

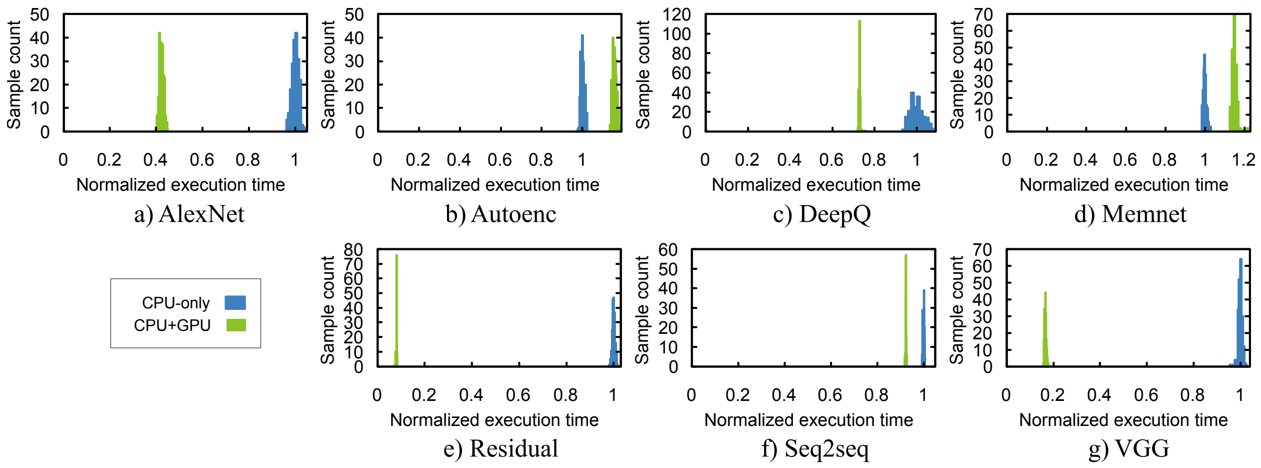

Figure 4: Execution time variance of the different samples of the various workloads.

recognition accuracy of their deep learning algorithms, considering that already in 2014 90% of the ImageNet teams used GPUs [28]. At the same time, we perceive that even though the GPU measurements also experience variability, they are consistently lower than their CPU-only counterparts across the various workloads, given that the code executed in the GPU is not affected by the operative system noise to the same extent as the CPU [29].

Table 1: GPU computation and memory usage efficiency.

Workload	Kernels execution time / API call time	Memcpy kernels time (%)
AlexNet	0.4298	3.2684
Autoenc	0.0010	15.9910
DeepQ	0.2585	11.1250
Memnet	0.0804	2.0889
Residual	0.8783	0.4036
Seq2seq	0.1911	0.9066
VGG	0.6278	1.4992

Observing Table 1, we can further analyze this behavior. When running on GPU and inspecting the *nvprof* reports, we find out that both Residual and VGG have 87.8% and 62.8% kernel efficiencies relative to the total CPU time used by the API calls, respectively, with data transfers between CPU and GPU taking less than 1.5% of the total kernel execution time in both cases. AlexNet, meanwhile, has almost 43% kernel usage efficiency increasing the data transfer overhead to about 3.25%, whereas Autoenc and Memnet do not benefit from using the GPU, consistent with the lack of parallel efficiency discussed in Section 4.3. Indeed, the performance of Autoenc and Memnet is compromised as a result of the GPU initialization, the data transfer and memory release overheads that represents more than 50% of the CPU time used by the GPU API calls. Autoenc, in particular, given its very low GPU usage measured in just microseconds obtains the lowest relative performance as a consequence of the high overhead produced by *cudaFree*, *cudaLaunch* and *cuDevicePrimaryCtxRetain* that represents almost the 100% of the API overhead.

4.5. Workloads breakdown

Table 2 presents the execution time breakdown of the different sections identified in the Fathom workloads normalized to the average total execution time (*wall time*) of each of them in CPU-only mode when running with the default Fathom settings.

This information supports the observations discussed in Section 4.4: workloads have to be sufficiently computational demanding to fully exploit the benefits of the Minsky platform so that performance is not significantly affected by the Python, TensorFlow and GPUs initialization overheads which, albeit small, represents almost 61% of the run time of the smallest Fathom models. This overhead, however, is inherent of the TensorFlow framework, despite the fact of being relatively insignificant when compared with the wall time of the higher computational demanding workloads as Residual and VGG.

We can observe that the time used by the *init* defined section remains stable and it is not benefited or harmed by the GPU utilization. The run time, on the other hand, as expected, is where we can recognize how the GPU-accelerated computing allow us to decrease the time spent by offloading the most compute-intensive

Table 2: **Normalized execution time for the sections identified in the Fathom workloads.**

Workload Section	AlexNet		Autoenc		DeepQ		Memnet	
	CPU Only	CPU + GPU	CPU Only	CPU + GPU	CPU Only	CPU + GPU	CPU Only	CPU + GPU
INIT	0.0392	0.0392	0.0583	0.0584	0.1629	0.1629	0.2583	0.2582
SETUP	0.0335	0.0305	0.0416	0.1312	0.0574	0.0762	0.0884	0.1699
RUN	0.8651	0.2880	0.2961	0.3414	0.5859	0.2890	0.1252	0.1758
MAIN	0.9391	0.3596	0.3965	0.5318	0.8068	0.5288	0.4733	0.6055
WALL TIME	1.000	0.4224	1.0000	1.1558	1.0000	0.7287	1.0000	1.1505

Workload Section	Residual		Seq2seq		VGG	
	CPU Only	CPU + GPU	CPU Only	CPU + GPU	CPU Only	CPU + GPU
INIT	0.0179	0.0179	0.6159	0.6164	0.0226	0.0226
SETUP	0.0088	0.0102	0.1848	0.1939	0.0222	0.0181
RUN	0.9635	0.0428	0.1742	0.0863	0.9288	0.0976
MAIN	0.9906	0.0713	0.9789	0.9008	0.9743	0.1389
WALL TIME	1.0000	0.0809	1.0000	0.9237	1.0000	0.1658

portions in those workloads that, as discussed above, have a sufficient kernel usage efficiency as a result of their internal structure.

4.6. Performance similarity

One interesting observation that results from the study presented in this work is the relationship between the scalability of the Fathom workloads in CPU-only mode and the performance benefits that they get when they run in CPU+GPU mode. Specifically, we observe that the use of the NVIDIA Tesla P100 GPU provides relatively larger benefits for workloads with good CPU scalability (like AlexNet, Residual and VGG). The rationale behind this observation is that GPU utilization usually increases with higher throughout of the master thread(s) running on the CPU side, which occurs when the internal structure of the workload is amenable to parallelization. In other words, the performance exhibited by the studied deep learning models is intrinsically tied to their application-level structure.

The level of GPU support for the operations that the framework exposes to the Fathom workloads is also crucial the fallback behavior is to run unsupported operations on the CPU consequently affecting the potential performance benefits. Therefore, the proper parallelization of deep learning workloads (like the Fathom applications studied in this paper) is key to fully exploit the advantages of high-throughput hybrid CPU-GPU systems like the Minsky platform used in this work.

5. Conclusions

This paper characterizes the Fathom benchmark suite running on a last-generation IBM POWER8 system with NVIDIA Tesla P100 GPUs and NVLink interconnects (“Minsky”). The work provides a quantitative analysis of the computational behavior of this platform when it executes on CPU-only and CPU+GPU modes.

We observe that the performance of deep learning algorithms is determined by the structure of the model at an application level. Specifically, an efficient use of the underlying hardware depends on the scalability exhibited by the model and the deep learning framework, as well as their capability to effectively offload execution to the attached GPU accelerator(s). Similarly, the models that have an easily parallelizable structure take great advantage of the Minsky platform. The performance efficiency of models whose internal structure is not amenable to parallelization drops quickly even in CPU-only mode. In addition, this study can provide insightful guidelines for the design of effective deep learning accelerators, as the ones being pursued within the DARPA-sponsored PERFECT project.

6. Acknowledgment

We are grateful to the members of the IBM-led project titled “Efficient Resilience in Embedded Computing,” sponsored by DARPA under its PERFECT program. An special mention to Dr. Pradip Bose and Dr. Alper Buyuktosunoglu from the ‘Efficient & Resilient Systems’ group at IBM T. J. Watson Research Center.

7. References

[1] J. Dean, "Large-scale deep learning for intelligent computer systems," BayLearn keynote speech, 2015.

[2] N. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, et al., "In-datacenter performance analysis of a tensor processing unit," ArXiv preprint arXiv:1704.04760, 2017.

[3] NVIDIA Corporation, DGX-1 deep learning system datasheet, Apr. 2016.

[4] IBM Corporation, IBM Power System S822LC for high performance computing datasheet, Mar. 2017.

[5] Power efficiency revolution for embedded computing technologies (PERFECT), https://www.darpa.mil/program/power-efficiency-revolution-for-embedded-computing-technologies.

[6] R. Adolf, S. Rama, B. Reagen, G. Wei, and D. M. Brooks, "Fathom: Reference workloads for modern deep learning methods," vol. abs/1608.06581, 2016. [Online]. Available: http://arxiv.org/abs/1608.06581.

[7] NVIDIA Corporation. (2017). Developing for OpenPOWER and NVIDIA NVLink, [Online]. Available: https://developer.nvidia.com/openpower (visited on 05/22/2017).

[8] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, et al., "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," ArXiv preprint arXiv:1603.04467, 2016.

[9] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in Proceedings of the 25th International Conference on Neural Information Processing Systems, ser. NIPS'12, 2012, pp. 1097–1105.

[10] D. P. Kingma and M. Welling, "Stochastic gradient VB and the variational auto-encoder," in Proceedings of the 2nd International Conference on Learning Representations, ser. ICLR'14, 2014.

[11] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," Science, vol. 313, no. 5786, pp. 504–507, 2006.

[12] A. B. Caldeira, V. Haug, and S. Vetter, IBM POWER SYSTEM S822LC for High Performance Computing Introduction and Technical Overview, 1st ed. IBM Redbooks, Oct. 2016, p. 9. [Online]. Available: http://www.redbooks.ibm.com/redpapers/pdfs/redp5405.pdf.

[13] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari with deep reinforcement learning," ArXiv preprint arXiv:1312.5602, 2013.

[14] J. Weston, S. Chopra, and A. Bordes, "Memory networks," ArXiv preprint arXiv:1410.3916, 2014.

[15] S. Sukhbaatar, A. Szlam, J. Weston, and R. Fergus, "End-to-end memory networks," in Proceedings of the 28th International Conference on Neural Information Processing Systems, ser. NIPS'15, 2015, pp. 2440–2448.

[16] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in Proceedings of the 27th International Conference on Neural Information Processing Systems, ser. NIPS'14, 2014, pp. 3104–3112.

[17] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in Proceedings of the 2016 IEEE conference on computer vision and pattern recognition, ser. CVPR'16, 2016, pp. 770–778.

[18] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," ArXiv preprint arXiv:1409.1556, 2014.

[19] A. Hannun, C. Case, J. Casper, B. Catanzaro, G. Diamos, E. Elsen, R. Prenger, S. Satheesh, S. Sengupta, A. Coates, et al., "Deep speech: Scaling up end-to-end speech recognition," ArXiv preprint arXiv:1412.5567, 2014.

[20] B. Sinharoy, J. Van Norstrand, R. J. Eickemeyer, H. Q. Le, J. Leenstra, D. Q. Nguyen, B. Konigsburg, K. Ward, M. Brown, J. E. Moreira, et al., "IBM POWER8 processor core microarchitecture," IBM Journal of Research and Development, vol. 59, no. 1, 2015.

[21] NVIDIA Corporation, Tesla P100 GPU accelerator datasheet, Oct. 2016.

[22] R. Adolf. (May 2017). Merge pull request #27 from zzzoom/tf-1.0.x, [Online]. Available: https://github.com/rdadolf/fathom/commit/9451f3.

[23] Python Software Foundation. (Aug. 2007). Timeit - measure execution time of small code snippets, [Online]. Available: https://docs.python.org/2/library/timeit.html.

[24] OpenMP Architecture Review Board, OpenMP application programming interface, Nov. 2015. [Online]. Available: http://www.openmp.org/wp-content/uploads/openmp-4.5.pdf.

[25] J. L. Hennessy and D. A. Patterson, Computer architecture: A quantitative approach, 4th. Morgan Kaufmann Publishers, 2011.

[26] J. S. Garofolo, L. F. Lamel, W. M. Fisher, J. G. Fiscus, D. S. Pallett, N. L. Dahlgren, and V. Zue, "TIMIT acoustic-phonetic continuous speech corpus," Linguistic data consortium, vol. 10, no. 5, 1993.

[27] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in Proceedings of the 2009 IEEE Conference on Computer Vision and Pattern Recognition, ser. CVPR'17, 2009, pp. 248–255.

[28] S. Jones. (Sep. 2014). NVIDIA GPUs power deep-learning winners in world cup of image recognition, [Online]. Available: https://blogs.nvidia.com/blog/2014/09/07/imagenet/.

[29] P. Beckman, K. Iskra, K. Yoshii, S. Coghlan, and A. Nataraj, "Benchmarking the effects of operating system interference on extreme-scale parallel machines," Cluster Computing, vol. 11, no. 1, pp. 3–16, 2008.