DNN Training Performance Analysis: A Divide and Conquer Approach

Anand Jayarajan
anandj@cs.toronto.edu
University of Toronto
Toronto, Ontario, Canada

1 Introduction

Designing hardware accelerators for DNN training is an active area of research right now [10, 12, 22]. These accelerators implement variety of optimizations specially tailored for DNN training. Robust and error-tolerant nature of the DNN training algorithms enables hardware researchers to explore optimizations involving approximate computations. Optimizations like reduced precision training [14, 20] or sparsity-aware training [22] can help better saturate hardware units and improve training throughput, but can delay the model from converging. Therefore, proper performance comparison of two different hardware optimizations must account for not only the raw performance numbers like training throughput (number of data samples processed per iteration), but their statistical efficiency as well.

The current standard practice for DNN performance comparison is to use Time-To-Accuracy (TTA) as the main metric. TTA is defined as the end-to-end training time to achieve a specific validation accuracy, usually set to the state-of-the-art result. Even though this metric has shown to be fairly robust [4], measuring TTA requires training benchmark models end-to-end and is time consuming. Table 1 shows a few standard DNN benchmarks and their corresponding training time on Nvidia P100 [16] reported by MLPerf [15]. Even with a powerful GPU like P100, certain benchmarks take weeks to achieve the state-of-the-art accuracy. Measuring TTA for hardware optimizations becomes especially challenging as most of them are prototyped on software-based simulators which are $6-7$ orders of magnitude slower compared to the real hardware [5]. Alternatively, prototyping optimizations on FPGAs or ASICs can also be time consuming, expensive and error prone. In this work, we propose a fast and affordable methodology for prototyping and analysing the performance of hardware optimizations for DNN training. We believe our new methodology and performance analysis tools can improve the productivity of the hardware researchers.

Benchmark	Training Time on Nvidia P100 (hours)
ResNet-50 [8]	147
MaskRCNN [7]	83
Seq2Seq [9]	18
MiniGo [18]	73

Table 1. MLPerf [15] training benchmark results v0.6

1.1 Divide and conquer approach

Towards the goal of this work, we make two key observations. The DNN training is an iterative process with each iteration having fairly similar performance characteristics. Thus estimating the training throughput require only a few representative measurements from the entire training period. On the other hand, estimating the statistical efficiency still need the models to be trained for a longer period of time, but does not require the optimizations to be implemented in circuit-level detail as is done in simulators. Benchmarking only parts of the optimization which could affect the numerical precision is sufficient to estimate the statistical efficiency. Based on these observations, we divide the process of performance analysis into two phases: First, the **simulation phase** which accurately simulates the hardware optimization on a simulator and runs the benchmarks for a few iterations to measure the average time taken to run each iteration. The fact that we only need a few iterations of training makes this phase affordable to run on slow simulators. Second, the **emulation phase**, which emulates the proposed optimization on a programmable general purpose hardware accelerator like GPU for measuring the convergence accuracy. The emulation phase measures the number of iterations required to reach the state-of-the-art validation accuracy. Measuring the convergence accuracy by emulating the optimization on fast GPUs makes the measurement process much faster than running the same on simulators or prototyping on FPGAs. The measurements from the simulation phase and emulation phase can then later be combined to produce standard metrics like TTA. We call this methodology the divide-and-conquer approach.

2 Contributions

- We propose a fast and affordable methodology for prototyping and analysing the performance of hardware designs/optimizations for DNN training.
- We provide an emulation tool for quickly prototype and measuring statistical efficiency of new hardware optimizations.
- We demonstrate the flexibility of our emulation tool by emulating custom operator implementation and custom data types.

3 Use Cases

There are a wide range of optimizations proposed for DNN training from both systems and architecture community. Therefore, it is challenging to come up with a generic method and tool for their performance analysis. We tackle this problem by targeting specific use cases. In the subsequent sections, we describe three commonly used optimization techniques and propose separate methods and tools for each of them.

Currently, our primary focus is to provide emulation tools as it is non-trivial for systems and architecture researchers to accurately analyze and estimate the statistical efficiency of an optimization. Moreover, accurate hardware simulators like GPGPU-Sim [13] and gem5 [2] already exist and is widely used in the community.

3.1 Use Case 1: Custom Operators

DNNs contain complex operators like matrix multiplication and convolution functions. Although many of these operators can be efficiently scaled on hardware accelerators, others like activation functions (e.g.: sigmoid, tanh) are harder to optimize and implement on custom hardwares because of their non-linear nature. Recent works [6, 21] have proposed approximate versions of these operators in order to make it easier to implement on FPGAs and ASICs with some tradeoffs on the model accuracy.

Even though most of the popular machine learning frameworks like TensorFlow [1] and PyTorch [11] have provision to incorporate custom operators, we experience that correctly implementing new operators is challenging especially if we need to implement their corresponding gradient operators as well. We find that using domain specific languages (DSL) provided by the machine learning compilers like TVM [3], Tensor Comprehension [19] and PlaidML [17] are much easier to write custom kernels and operators because of their concise syntax that resemble more familiar mathematical notations like Einstein Summation. Additionally, PlaidML can automatically infer the gradient operator from the DSL code. We extend the PlaidML DSL to support programming primitives like conditional statements and switch-cases. Using the extended DSL, we were able to successfully emulate an approximate version of $tanh()$ activation operator.

3.2 Use Case 2: Custom Data Formats

Reduced precision training [14] is one of the most explored optimizations in hardware side. Some of these optimizations require the use of non-standard floating point representations or fixed point representation which are currently not supported by mainstream hardware accelerators. For example, Wang et al. [20] recently proposed 8-bit precision training which require $fp8$ with a (sign, exponent, mantissa) format of $(1, 5, 2)$ bits and $fp16$ with a $(1, 6, 9)$ format. In order to emulate such custom types, we have extended PlaidML compiler to support configurable data formats. With our tool, users only need to provide the specifications of the new data type like (sign, exponent, mantissa) bits for floating-point or (integer, decimal) bits for fixed-points along with implementations of basic primitive operations on the new data type as shown in the Listing 1.

```
typedef struct {
    uint s : 1;
    uint e : 5;
    uint m : 2;
} new_fp;
new_fp convert(float f);
float as_float(new_fp nf);
new_fp add(new_fp a, new_fp b);
new_fp log(new_fp a);
```

Listing 1. Custom data type specification

3.3 Use Case 3: Custom processing unit emulation

Custom processing unit emulation can emulate different hardware computation units which run computations on the newly defined data type. Listing 2 shows an example definition of a custom MAD unit which accepts two *new_fp* typed arguments in vectorized format of size 4 and returns the element-wise output.

```
new_fp4 mad(new_fp4 a, new_fp4 b);
```

Listing 2. Custom vectorized MAD unit emulation

Our emulator automatically uses such custom implementations to execute corresponding operations in the training process.

4 Conclusion

In this work, we propose a fast and affordable performance analysis methodology called divide-and-conquer approach for DNN training hardware optimizations. We additionally provide a proposal for an emulation tool which can easily and accurately estimate the statistical efficiency of a wide range of hardware optimizations. We propose a machine learning compiler-based emulator on top PlaidML [17] and demonstrate its flexibility on emulating custom data types, custom operator implementations and custom processing units.

References

[1] Martin Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2016. TensorFlow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. 265–283. https://www.usenix.org/system/files/conference/osdi16/osdi16-abadi.pdf

[2] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood. 2011. The Gem5 Simulator. *SIGARCH Comput. Archit. News* 39, 2 (Aug. 2011), 1–7. https://doi.org/10.1145/2024716.2024718

[3] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Haichen Shen, Meghan Cowan, Leyuan Wang, Yuwei Hu, Luis Ceze, Carlos Guestrin, and Arvind Krishnamurthy. 2018. TVM: An Automated End-to-End Optimizing Compiler for Deep Learning. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. USENIX Association, Carlsbad, CA, 578–594. https://www.usenix.org/conference/osdi18/presentation/chen

[4] Cody Coleman, Daniel Kang, Deepak Narayanan, Luigi Nardi, Tian Zhao, Jian Zhang, Peter Bailis, Kunle Olukotun, Christopher Ré, and Matei Zaharia. 2018. Analysis of DAWNBench, a Time-to-Accuracy Machine Learning Performance Benchmark. *CoRR* abs/1806.01427 (2018). arXiv:1806.01427 http://arxiv.org/abs/1806.01427

[5] David Kaplan. 2015. When hardware must just work. https://youtu.be/e2vPp0fQUkM?t=637

[6] J. Fowers, K. Ovtcharov, M. Papamichael, T. Massengill, M. Liu, D. Lo, S. Alkalay, M. Haselman, L. Adams, M. Ghandi, S. Heil, P. Patel, A. Sapek, G. Weisz, L. Woods, S. Lanka, S. K. Reinhardt, A. M. Caulfield, E. S. Chung, and D. Burger. 2018. A Configurable Cloud-Scale DNN Processor for Real-Time AI. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. 1–14. https://doi.org/10.1109/ISCA.2018.00012

[7] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross B. Girshick. 2017. Mask R-CNN. *CoRR* abs/1703.06870 (2017). arXiv:1703.06870 http://arxiv.org/abs/1703.06870

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep Residual Learning for Image Recognition. *CoRR* abs/1512.03385 (2015). arXiv:1512.03385 http://arxiv.org/abs/1512.03385

[9] Felix Hieber, Tobias Domhan, Michael Denkowski, David Vilar, Artem Sokolov, Ann Clifton, and Matt Post. 2017. Sockeye: A Toolkit for Neural Machine Translation. *CoRR* abs/1712.05690 (2017). arXiv:1712.05690 http://arxiv.org/abs/1712.05690

[10] Mohsen Imani, Saransh Gupta, Yeseong Kim, and Tajana Rosing. 2019. FloatPIM: In-memory Acceleration of Deep Neural Network Training with High Precision. In *Proceedings of the 46th International Symposium on Computer Architecture (ISCA '19)*. ACM, New York, NY, USA, 802–815. https://doi.org/10.1145/3307650.3322237

[11] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. 2014. Caffe: Convolutional Architecture for Fast Feature Embedding. In *Proceedings of the 22Nd ACM International Conference on Multimedia (MM '14)*. ACM, New York, NY, USA, 675–678. https://doi.org/10.1145/2647868.2654889

[12] Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre-luc Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, Matt Dau, Jeffrey Dean, Ben Gelb, Tara Vazir Ghaemmaghami, Rajendra Gottipati, William Gulland, Robert Hagmann, C. Richard Ho, Doug Hogberg, John Hu, Robert Hundt, Dan Hurt, Julian Ibarz, Aaron Jaffey, Alek Jaworski, Alexander Kaplan, Harshit Khaitan, Daniel Killebrew, Andy Koch, Naveen Kumar, Steve Lacy, James Laudon, James Law, Diemthu Le, Chris Leary, Zhuyuan Liu, Kyle Lucke, Alan Lundin, Gordon MacKean, Adriana Maggiore, Maire Mahony, Kieran Miller, Rahul Nagarajan, Ravi Narayanaswami, Ray Ni, Kathy Nix, Thomas Norrie, Mark Omernick, Narayana Penukonda, Andy Phelps, Jonathan Ross, Matt Ross, Amir Salek, Emad Samadiani, Chris Severn, Gregory Sizikov, Matthew Snelham, Jed Souter, Dan Steinberg, Andy Swing, Mercedes Tan, Gregory Thorson, Bo Tian, Horia Toma, Erick Tuttle, Vijay Vasudevan, Richard Walter, Walter Wang, Eric Wilcox, and Doe Hyun Yoon. 2017. In-Datacenter Performance Analysis of a Tensor Processing Unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture (ISCA '17)*. ACM, New York, NY, USA, 1–12. https://doi.org/10.1145/3079856.3080246

[13] Jingwen Leng, Tayler Hetherington, Ahmed ElTantawy, Syed Gilani, Nam Sung Kim, Tor M. Aamodt, and Vijay Janapa Reddi. 2013. GPUWattch: Enabling Energy Optimizations in GPGPUs. *SIGARCH Comput. Archit. News* 41, 3 (June 2013), 487–498. https://doi.org/10.1145/2508148.2485964

[14] Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory F. Diamos, Erich Elsen, David García, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, and Hao Wu. 2017. Mixed Precision Training. *CoRR* abs/1710.03740 (2017). arXiv:1710.03740 http://arxiv.org/abs/1710.03740

[15] MLPerf. 2019. MLPerf Training v0.6 Results. https://mlperf.org/training-results-0-6/

[16] Nvidia. 2017. Nvidia Tesla P100. https://www.nvidia.com/en-us/data-center/tesla-p100/

[17] PlaidML. 2017. Intel AI PlaidML. https://vertexai-plaidml.readthedocs-hosted.com/en/latest/index.html

[18] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. 2017. Mastering the game of Go without human knowledge. *Nature* 550 (Oct. 2017), 354–. http://dx.doi.org/10.1038/nature24270

[19] Nicolas Vasilache, Oleksandr Zinenko, Theodoros Theodoridis, Priya Goyal, Zachary DeVito, William S. Moses, Sven Verdoolaege, Andrew Adams, and Albert Cohen. 2018. Tensor Comprehensions: Framework-Agnostic High-Performance Machine Learning Abstractions. *CoRR* abs/1802.04730 (2018). arXiv:1802.04730 http://arxiv.org/abs/1802.04730

[20] Naigang Wang, Jungwook Choi, Daniel Brand, Chia-Yu Chen, and Kailash Gopalakrishnan. 2018. Training Deep Neural Networks with 8-bit Floating Point Numbers. *CoRR* abs/1812.08011 (2018). arXiv:1812.08011 http://arxiv.org/abs/1812.08011

[21] Shuo Wang, Zhe Li, Caiwen Ding, Bo Yuan, Yanzhi Wang, Qinru Qiu, and Yun Liang. 2018. C-LSTM: Enabling Efficient LSTM using Structured Compression Techniques on FPGAs. *CoRR* abs/1803.06305 (2018). arXiv:1803.06305 http://arxiv.org/abs/1803.06305

[22] Jiaqi Zhang, Xiangru Chen, Mingcong Song, and Tao Li. 2019. Eager Pruning: Algorithm and Architecture Support for Fast Training of Deep Neural Networks. In *Proceedings of the 46th International Symposium on Computer Architecture (ISCA '19)*. ACM, New York, NY, USA, 292–303. https://doi.org/10.1145/3307650.3322263