FASE: A Fast, Accurate and Seamless Emulator for Custom Numerical Formats

John Osorio*, Adrià Armejach*, Eric Petit‡, Greg Henry‡ and Marc Casas*
*Barcelona Supercomputing Center (BSC) and Universitat Politècnica de Catalunya (UPC), Barcelona, Spain
‡Intel, Oregon, USA
Email: john.osorio@bsc.es

***Abstract*—Deep Neural Networks (DNNs) have become ubiquitous in a wide range of application domains. Despite their success, training DNNs is an expensive task that has motivated the use of reduced numerical precision formats to improve performance and reduce power consumption. Emulation techniques are a good fit to understand the properties of new numerical formats on a particular workload. However, current SoA techniques are not able to perform these tasks quickly and accurately on a wide variety of workloads.**

We propose FASE, a Fast, Accurate, and Seamless Emulator that leverages dynamic binary translation to enable emulation of custom numerical formats. FASE is *fast*: allowing emulation of large unmodified workloads; *accurate*: emulating at the instruction operand level; and *seamless*: as it does not require any code modifications and works on any application or DNN framework without any language, compiler, or source code access restrictions.

I. INTRODUCTION

Current trends on DNNs indicate that training costs will continue to grow as SoA DNNs feature increasingly large parameter counts [1]. There are already approaches on reducing the training computation costs via mechanisms that incur accuracy degradations [2]–[4]. Additionally, there are approaches able to reduce training costs without reducing DNNs accuracy. These approaches rely on reduced computer number formats [5]–[8]. To decide among all possible potential format designs which ones display the best opportunities for efficient and accurate DNN training, it is critical to empirically evaluate them with as much fidelity as possible and on as much real neural net topology and real input datasets as possible. The emulation of these reduced precision approaches becomes one of the most important and costly phases to evaluate the reliability of new numerical data types. The emulation helps to avoid cost overrun, by avoiding costly hardware implementations.

There are various SoA techniques to emulate reduced precision approaches. RPE [9] emulates the use of low precision approaches in numerical simulations based on type overloading at source level. RPE achieves emulation latencies of around $40\times$ with respect to native executions on real hardware, but does not support the large Python frameworks used in ML. Furthermore, source level instrumentation might interact with the compiler's ability to optimize the code and therefore the result of the emulation might by inaccurate in the context of evaluating hardware design. TensorQuant [10], proposes two source level approaches, intrinsic (fine-grain) and extrinsic (coarse-grain), to emulate low precision using Tensorflow. The extrinsic approach is an approximation where the rounding process is done just on high level operators like convolutions. This is the mode implemented in QPyTorch [11] to address the PyTorch framework. The intrinsic approach rounds each individual floating point operation and displays a latency of $50\times$ with respect to native executions. It is a source level approach that can be used to evaluate all implementation of neural network based on Tensorflow. All of these approaches are designed targeting specific DNN frameworks and require changes on the framework and model source code.Other tools like Verificarlo [12] work at the compiler level, and can be applied to any Python framework. However, they do require complex recompilation and debugging of the library and user code.

To overcome these issues we propose FASE: a fast, accurate and seamless tool that enables the emulation of custom numerical formats on any application. FASE relies on dynamic binary instrumentation using PIN [13] to perform fine-grain instruction-level instrumentation. In addition, FASE seamlessly works on any application or DNN framework without any language, compiler or source access restrictions. Since no code modification or recompilation steps are necessary, FASE guarantees that the instrumented binary matches the original one. Therefore, FASE works on all DNN frameworks, such as: Caffe [14], Tensorflow [15] and Pytorch [16]. While fine-grain instrumentation can inject large latencies, FASE have latencies that range from $17\times$ to $39\times$, which are comparable to other fine-grain SoA techniques. As a result, FASE enables hardware architects to understand numerical behaviour before committing to costly hardware implementations.

This poster makes the following contributions:

- We propose FASE, an emulation tool for arbitrary numerical formats that enables accurate emulation of large workloads without requiring any source code modifications or access to third-party dynamically linked libraries.
- We outline the design and implementation details, that enable accurate emulation while keeping the injected latencies low for feasible large-scale experimentation.
- An exhaustive evaluation campaign that demonstrates that FASE works on large-scale experiments using multiple numerical formats.

 DOI 10.1109/ISPASS55109.2022.00017

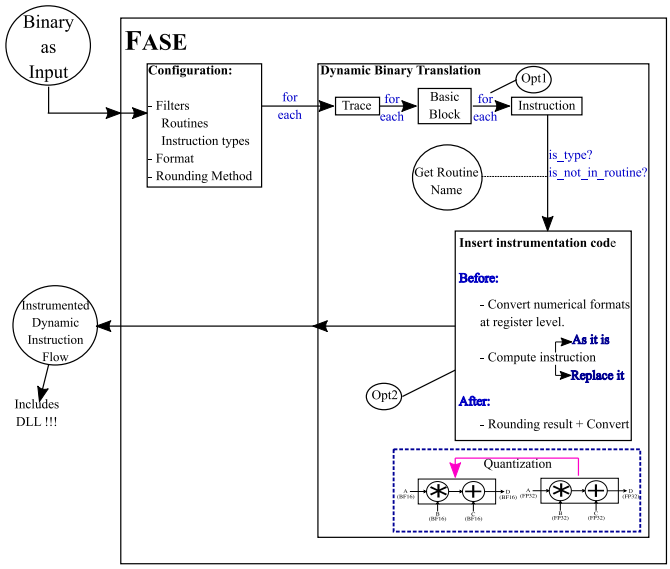

Fig. 1: FASE Implementation Overview

II. FASE Design

Our goal is to design FASE with simplicity in mind by enabling fast, accurate and seamless emulation of reduced precision formats. In addition, we want our tool to be able to emulate code of external dynamically linked libraries, as many applications rely on such libraries which contain key optimized routines.

FASE aims to provide an accurate and seamless method. To achieve a fine-grain emulation, we propose to leave the target application unmodified and operate at binary level intercepting the executed machine instruction. By identifying key floating-point instructions, for which we can modify the input and output operands, FASE can seamlessly work on any application and DNN framework including dynamically linked external libraries.

III. FASE Implementation

In order to provide a fast, accurate and seamless experience; FASE relies on dynamic binary translation (DBT). DBT enables modifications in the dynamic instruction flow of any application binary, as well as on any dynamically linked libraries the binary invokes. These modifications are done during the *instrumentation* step, which is executed once.

Figure 1 shows an overview of the DBT instrumentation step on FASE. FASE can be attached to any binary, and is configured through a simple configuration file that specifies the desired instrumentation parameters in terms of routines and instructions to be instrumented as well as the emulated reduced precision format and rounding method. The DBT step which performs the instrumentation goes through each statically defined basic block once, and for each instruction it can insert instrumentation code. In our context, for each instruction, we want to perform up to three code insertions:

1) **Before:** Insert code that converts the source registers of the instruction to the desired reduced precision format and applies the desired rounding.

TABLE I: Large-scale experiments using FASE

Model	Dataset	Accuracy			
		FP32	BF16	MP	BF16x2
ResNet18	CIFAR100	71.91%	71.46%	71.89%	71.95%
ResNet34	CIFAR100	73.21%	72.83%	73.86%	72.66%
ResNet50	CIFAR100	74.78%	69.24%	74.25%	72.57%
ResNet101	CIFAR100	75.93%	67.10%	75.65%	76.00%
MobileNetV2	CIFAR100	75.04%	73.92%	75.16%	74.82%
AlexNet	ImageNet	60.79%	57.80%	60.18%	N/A
Inception	ImageNet	74.01%	72.03%	73.73%	N/A
LSTMx2 (Perplexity)	PTB	86.86	137.69	87.09	86.90
Transformers (BLEU)	IWSLT16	34.53	34.86	34.66	34.65

2) **Instruction:** In most cases the instruction can be executed as is with the modified source registers. In some cases, when the numerical format will not execute as expected on the existing instruction or available hardware, the instruction needs to be replaced by equivalent code that emulates the intended behaviour.
3) **After:** Insert code that converts the output to the desired reduced precision format and applies the rounding mechanism.

IV. FASE evaluation

Our experimental methodology considers the evaluation of FASE on DNN frameworks like Caffe and PyTorch. Our experiments are performed on an Intel Xeon Platinum 8160 processors. We compile each framework from source enabling AVX512 Intel optimizations on all of them.

To show FASE supports real workloads we perform a set of experiments. These tests consider the use of several DNN models, datasets and numerical datatypes. We report the validation accuracy after training, BLEU Score, or perplexity depending on the workload type. We compare the obtained accuracies against the reference implementation using FP32. We use FASE to emulate three different numerical formats in order to demonstrate the versatility of our tool:

- **BF16** with RNE rounding used until now.
- The mixed-precision (**MP**) [17], [18] approach that employs FP32 precision in batch normalization and weight update layers. And performs FMA instructions using BF16 source inputs for the multiplication and an FP32 input for the accumulator, returning an FP32 value as output.
- A compound datatype that represents FP32 values using a tuple of BF16 values (**BF16x2**) [19]. Note that this format requires changing the original instruction with ad-hoc code that performs the operation using the BF16x2 format.

We consider several DNN models as you see in Table I. We use PyTorch for the object classification on CIFAR and sequence models. The remaining experiments use Caffe. Table I shows the results of using FASE for several full DNN training workloads. We compare the accuracy of each network using our tool emulating different numerical formats (BF16, MP and BF16x2), and the FP32 native execution.

With FASE we can determine if a reduced precision format is able to achieve the desired level of accuracy. This set of results illustrates the potential of FASE to emulate different

numerical formats and to extract conclusions on their applicability. FASE can also be employed to study scenarios where numerical precision is changed at runtime depending on application progress, and to study other custom floating-point representations; making it a compelling fast, accurate and seamless tool.

ACKNOWLEDGMENT

Marc Casas has been partially supported by the Grant RYC-2017-23269 funded by MCIN/AEI/ 10.13039/501100011033 and by "ESF Investing in your future". Adrià Armejach is a Serra Húnter Fellow and has been partially supported by the Grant IJCI-2017-33945 funded by MCIN/AEI/ 10.13039/501100011033. John Osorio has been partially supported by the Grant PRE2019- 090406 funded by MCIN/AEI/ 10.13039/501100011033 and by "ESF Investing in your future". This work has been partially supported by Intel under the BSC-Intel collaboration and European Union's Horizon 2020 research and innovation programme under grant agreement No 955606 - DEEP-SEA EU project.

REFERENCES

[1] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," 2020.

[2] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning convolutional neural networks for resource efficient inference," 2017.

[3] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," 2018.

[4] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," 2019.

[5] Y. Fu, H. You, Y. Zhao, Y. Wang, C. Li, K. Gopalakrishnan, Z. Wang, and Y. Lin, "Fractrain: Fractionally squeezing bit savings both temporally and spatially for efficient dnn training," in *Neurips*, 2020.

[6] Y. Fu, H. Guo, M. Li, X. Yang, Y. Ding, V. Chandra, and Y. Lin, "{CPT}: Efficient deep neural network training via cyclic precision," in *International Conference on Learning Representations*, 2021. [Online]. Available: https://openreview.net/forum?id=87ZwsaQNHPZ

[7] X. Sun, J. Choi, C. Y. Chen, N. Wang, S. Venkataramani, V. Srinivasan, X. Cui, W. Zhang, and K. Gopalakrishnan, "Hybrid 8-bit Floating Point (HFP8) Training and Inference for Deep Neural Networks," in *NeurIPS*, 2019.

[8] X. Sun, N. Wang, C.-Y. Chen, J.-M. N. Ankur, A. Xiaodong, C. Swagath, V. Kaoutar, E. Maghraoui, V. Srinivasan, and K. Gopalakrishnan, "Ultra-Low Precision 4-bit Training of Deep Neural Networks," in *Neurips*, 2020.

[9] A. Dawson and P. D. Düben, "rpe v5: an emulator for reduced floating-point precision in large numerical simulations," *Geoscientific Model Development*, vol. 10, no. 6, pp. 2221–2230, 2017. [Online]. Available: https://gmd.copernicus.org/articles/10/2221/2017/

[10] D. M. Loroch, N. Wehn, F.-J. Pfreundt, and J. Keuper, "Tensorquant - a simulation toolbox for deep neural network quantization," 2017.

[11] T. Zhang, Z. Lin, G. Yang, and C. D. Sa, "Qpytorch: A low-precision arithmetic simulation framework," 2019.

[12] Y. Chatelain, E. Petit, P. de Oliveira Castro, G. Lartigue, and D. Defour, "Automatic exploration of reduced floating-point representations in iterative methods," in *Euro-Par 2019 Parallel Processing - 25th International Conference*, ser. Lecture Notes in Computer Science. Springer, 2019.

[13] C. K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. J. Reddi, and K. Hazelwood, "Pin: Building customized program analysis tools with dynamic instrumentation," *ACM SIGPLAN Notices*, 2005.

[14] Intel. Intel caffe framework optimization. [Online]. Available: https://github.com/intel/caffe

[15] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mane, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viegas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems," 2016.

[16] A. Paszke, S. Gross, S. Chintala, and G. Chanan. (2020) Pytorch. [Online]. Available: https://github.com/pytorch/pytorch

[17] D. Kalamkar, D. Mudigere, N. Mellempudi, D. Das, K. Banerjee, S. Avancha, D. T. Vooturi, N. Jammalamadaka, J. Huang, H. Yuen, J. Yang, J. Park, A. Heinecke, E. Georganas, S. Srinivasan, A. Kundu, M. Smelyanskiy, B. Kaul, and P. Dubey, "A Study of BFLOAT16 for Deep Learning Training," 2019.

[18] P. Micikevicius, S. Narang, J. Alben, G. Diamos, E. Elsen, D. Garcia, B. Ginsburg, M. Houston, O. Kuchaiev, G. Venkatesh, and H. Wu, "Mixed Precision Training," *ICLR*, 2018.

[19] G. Henry, P. T. P. Tang, and A. Heinecke, "Leveraging the bfloat16 artificial intelligence datatype for higher-precision computations," 2019.