Short Paper: Accelerating Hyperparameter Optimization Algorithms with Mixed Precision

Marcel Aach
Jülich Supercomputing Centre
Jülich, Germany
University of Iceland
Reykjavík, Iceland

Rakesh Sarma
Jülich Supercomputing Centre
Jülich, Germany

Eray Inanc
Jülich Supercomputing Centre
Jülich, Germany

Morris Riedel
University of Iceland
Reykjavík, Iceland
Jülich Supercomputing Centre
Jülich, Germany

Andreas Lintermann
Jülich Supercomputing Centre
Jülich, Germany

ABSTRACT

Hyperparameter Optimization (HPO) of Neural Networks (NNs) is a computationally expensive procedure. On accelerators, such as NVIDIA Graphics Processing Units (GPUs) equipped with Tensor Cores, it is possible to speed-up the NN training by reducing the precision of some of the NN parameters, also referred to as mixed precision training. This paper investigates the performance of three popular HPO algorithms in terms of the achieved speed-up and model accuracy, utilizing early stopping, Bayesian, and genetic optimization approaches, in combination with mixed precision functionalities. The benchmarks are performed on 64 GPUs in parallel on three datasets: two from the vision and one from the Computational Fluid Dynamics domain. The results show that larger speed-ups can be achieved for mixed compared to full precision HPO if the checkpoint frequency is kept low. In addition to the reduced runtime, small gains in generalization performance on the test set are observed.

CCS CONCEPTS

• **Computing methodologies → Parallel algorithms**.

KEYWORDS

Hyperparameter Optimization, Mixed Precision, High-Performance Computing

1 INTRODUCTION

The performance of Machine Learning (ML) models is highly dependent on the choice of hyperparameters. The hyperparameter space is not only spanned by features of the ML architecture, such as the number of neurons or layers, but also by parameters of the optimizer, e.g., the learning rate, batch size, or regularization like weight decay. Hyperparameter Optimization (HPO) describes the systematic search process for well-performing combinations of these parameters. Since the different configurations (or "trials") are independent, the process is ideally suited to be exploited in parallel on High-Performance Computing (HPC) systems. This way, several models across different scientific domains have been improved [2, 19]. However, HPO is still computationally challenging, as the ML models and the datasets trained on are continuously growing in size. There is hence a great interest in reducing the computational complexity of HPO. From a software perspective, one method to reduce this complexity is using early stopping techniques, which terminate trials with bad performance before they are fully trained. Another approach is to apply techniques that more intelligently sample the hyperparameter space, e.g., to use Bayesian Optimization (BO) [7] or evolutionary methods [10]. From a hardware perspective, using advanced accelerators, such as Graphics Processing Units (GPUs) or Tensor Processing Units (TPUs), can drastically speed up the training process. Employing mixed precision arithmetics in training has intensively been investigated [16] and is frequently used in practice for deep Neural Networks (NNs) at large scale. The literature is lacking an evaluation of this GPU training feature for HPO tasks.

Therefore, this work aims to investigate Automatic Mixed Precision (AMP) as one of the improvements for HPO using modern NVIDIA GPUs, combined with three different HPO algorithms and a random search baseline. Half and full precision computations are mixed adaptively to run NN training more efficiently. The performance of the AMP approach is evaluated by comparing the corresponding results to those obtained with full precision arithmetics. Three datasets are used as basis for comparison: two vision datasets, i.e., cifar-10 [13] and ImageNet [17], and a Computational Fluid Dynamics (CFD) dataset [1]. The search space is defined by architectural parameters, such as the number of filters in a Convolutional Neural Network (CNN), and by optimizer-related parameters, e.g., the learning rate or momentum. The evaluations are performed on the GPU partition of the JURECA-DC machine at the Jülich Supercomputing Centre [12], using 64 NVIDIA A100 GPUs per HPO run in parallel. Code for reproducing this work is available on Gitlab[1].

The work is structured as follows. In Sec. 2 the background on HPO and the AMP package is described. Section 3 details the experimental setup, which is used for generating the results presented in Sec. 4. Finally, Sec. 5 provides the conclusion and future work.

[1]https://gitlab.jsc.fz-juelich.de/CoE-RAISE/FZJ/mixed-precision-hpo

2 BACKGROUND

2.1 Hyperparameter Optimization Methods

In mathematical notation, the performance of a NN with respect to a metric, e.g., the mean-squared error on a validation dataset, is described by a function $f : \mathcal{X} \to \mathbb{R}$, where $\mathcal{X}$ is the space of hyperparameters. The HPO process performs minimization of the objective function f by discovering combinations of hyperparameters $x^* \in \mathcal{X}$ such that $x^* \in \arg\min_{x \in \mathcal{X}} f(x)$. Evaluating the objective function is costly, because it requires the complete training of the NN, which is the main reason for HPO being compute intensive.

The acceleration of HPO frequently makes use of two approaches: (1) the evaluation of f on a smaller compute budget using "successive halving" [11] and (2) using better-informed hyperparameter combinations. In the first approach, all trials are run until a certain point in time (usually a few epochs), at which their performance is assessed. A fraction of the under-performing trials are terminated (early stopping), while continuing with the rest. HyperBand (HB) [14] was the first algorithm to implement this concept. Its successor, the Asynchronous Successive Halving Algorithm (ASHA) [15], is nowadays more frequently used. BO belongs to category (2) and approximates f with a probabilistic model, where new hyperparameter configurations are chosen based on the performance of earlier configurations. The Bayesian Optimization and HyperBand (BOHB) [7] algorithm combines both BO and early stopping. Furthermore, there exist genetic methods mimicking the process of evolution for finding optimal hyperparameters. First, an initial population of different ML models with randomly sampled hyperparameters is trained for a few epochs (a generation). Then, the performance is measured, and the models are ranked according to their results. Subsequently, different genetic operations, e.g., mutations, are applied. In the case of mutation, the worst performing trials copy the state and hyperparameters of the best performing models and apply small perturbations to these parameters. One of the most commonly used evolutionary HPO algorithm is Population Based Training (PBT) [10]. The iterative nature of the optimization process allows to find a series of hyperparameters (e.g., schedules).

2.2 Automatic Mixed Precision

Typically, computations and storage of deep NN parameters, e.g., in the forward and backward pass, use single (32 bit floating point or `FP32`) precision. However, with increasing model and dataset sizes, it is found that not all parameters require such high precision [16], leading to a potential reduction in computation time and disk space. One option is to apply half (16 bit floating point or `FP16`) precision, supported by GPUs, such as the NVIDIA A100 or AMD MI250. However, since values smaller than 2^{-24} cannot be represented in `FP16` and also just a few non-representable values (Not a Numbers – `NaNs` and Infinities – `Infs`) can break a training, a master copy of the weights in `FP32` is kept in memory. To detect non-representable values and to avoid the propagation and deterioration of the model accuracy, the AMP training workflow with gradient scaling exists (integrated naively into PyTorch), see Algo. 1.

Note that occasionally skipping the optimizer step does not impair the convergence rate [16]. Due to regularization effects from the lower precision, some models can even reach higher accuracy [4]. Also, according to NVIDIA, leveraging lower precision

Algorithm 1 PyTorch AMP Workflow Pseudocode

```
(1) Compute network forward pass in FP16
(2) Compute loss in FP32; scale by a large factor to ensure
    representability in FP16
(3) Compute backward pass in FP16 and check for NaN/Inf
(4) If result contains no NaN/Inf then
      (a) Unscale gradients and update optimizer
    Else
      (a) Skip optimizer update
```

computations results in large speed-ups. They compare the peak performance of the A100 GPU with Tensor Cores in half precision and achieve 312 TFLOPS while it is at 156 TFLOPS in full precision[2], i.e., a factor of two should be expected.

In the context of HPO, mixed precision computations have so far been mainly leveraged solely to find suitable network architectures for low energy inference on certain hardware [3]. They have not been used to accelerate the HPO process itself, which is what the present work investigates.

3 EXPERIMENTAL SETUP

3.1 High-Performance Computing System

The experiments are performed on the GPU partition of JURECA-DC featuring two AMD EPYC 7742 Central Processing Units (CPUs) with 128 cores @2.25 GHz and four NVIDIA A100 GPUs with each 40 GB HBM2e memory. For the experiments, `CUDA/11.7`, `PyTorch/2.0.1`, and `Ray Tune/2.6.2` is used. With the distributed computing library Ray Tune[3], hyperparameter trials can be distributed across different nodes and the NN training across multiple accelerators using data-parallel methods. For the vision and CFD datasets, four GPUs and two GPUs are used for each trial. In total, 64 GPUs are allocated for the complete HPO, resulting in 16/32 concurrent HPO evaluations. The number of trials for ASHA and BOHB is fixed to 128, for PBT and Random Search (RAND) to 16 and 32.

3.2 Datasets and Models

The two vision datasets for benchmarking are cifar-10 and ImageNet-1k. For both, a 80/20 split of the training set is used to generate a new validation set for selecting the best performing HPO trial (i.e., final model selection). The original validation sets are used as test sets to evaluate the final performance of the best model to rule out overfitting. As the cifar-10 dataset is too small in terms of image dimension to measure substantial speed-ups from `FP16` training, the images are upscaled to 225x255 pixels.

For cifar-10, the hyperparameter search space consists of a fixed CNN topology, where the number of convolutional filters are varied for each model stage, adapted from NATS-Bench [6]. Apart from the architectural parameters, the learning rate, weight decay, and the number of warm-up epochs are optimized, resulting in an eight-dimensional search space. In the PBT case, the learning rate and

[2]https://www.nvidia.com/en-us/data-center/a100/
[3]https://www.ray.io/

weight decay are adapted after each epoch, yielding a schedule of multiple learning rates and weight decay values.

ImageNet-1k uses a custom ResNet architecture [8], where the number of channels vary throughout the model. As a performance metric, the classification accuracy is used.

The CFD dataset holds turbulent boundary layer flow data from a simulation [1]. The ML model used is a Convolutional Autoencoder for flow reconstruction. It consists of an encoder, a latent space (i.e., a lower dimensional representation of the input), and a decoder. The encoder and decoder both feature four convolutional layers, where the first two layers perform down- and up-sampling to achieve the compression in the latent space, while the other two perform regular convolution. The model is taken from the AI4HPC repository [9], which offers a selection of ML application codes optimized for HPC. The model remains fixed and only optimizer-related parameters (learning rate, weight decay, momentum, Nesterov momentum, and warm-up epochs) define the search space. The difference between the input and the (reconstructed) output flow field is quantified using the mean squared error metric. Training, validation and test set are generated by splitting the original dataset time-wise.

4 RESULTS

The results of combining RAND, ASHA, BOHB, and PBT with AMP are reported in Tab. 1, averaged over three different splits of the training and validation set. Additionally, the speed-ups achieved by utilizing AMP over the full precision training are depicted in Fig. 1. It should be noted that in the CFD use case the model size remains constant, i.e., it is expected that each training epoch takes roughly the same amount of time. In the cifar-10 and ImageNet-1k cases the model size changes, as architectural parameters are part of the search space, i.e., the runtime per epoch might change from trial to trial. From Fig. 1 (blue bars) it is obvious that for the CFD model, using AMP with RAND and ASHA, yields with a factor of ≈ 1.3 in relative speed-up over full precision training. This result is expected as the model size remains constant and ASHA is essentially random search with early stopping. Benchmarking a single CFD model training run (w/o HPO) with full and mixed precision yields a factor of $^{2561s}/_{1766s} \approx 1.45$, which acts as an approximate upper speed-up limit. For the vision datasets, the attained speed-ups for ImageNet-1k are even higher with up to 1.56 for ASHA and RAND. However, they are much lower for cifar-10, which is at $1.07 - 1.13$. Larger and smaller models benefit differently from mixed precision training. In general, the ImageNet-1k models are one order of magnitude larger than the cifar-10 models. The benefits of AMP acceleration hence become notably larger and smaller, depending on the case.

A large speed-up on all datasets is observed for BOHB. Analyzing the HPO run output showed that BOHB performs early stopping much less aggressive than ASHA. Even though the same reduction factor for both algorithms is used (only the top 25% trials are allowed to continue), BOHB trains each configuration for at least three epochs while ASHA terminates many already after the first epoch. Starting a new trial is associated with an overhead that becomes (relatively) smaller when the trial is trained for longer time. For this reason a large speed-up is observed in the BOHB case of AMP compared to the full precision training. As can be seen in Tab. 1,

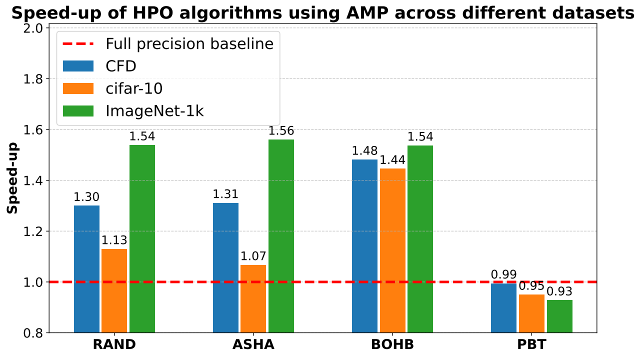

Figure 1: Experimental results of running RAND, ASHA, BOHB, and PBT with different datasets and search spaces.

this leads to much longer total runtimes, but also to sometimes better test set results.

No speed-up is achieved for the PBT algorithm – AMP even increases its runtime. The main reason for this is the way the PBT optimization works. The trials running concurrently are compared not only in terms of their performance on a metric, but also bad-performing trials copy the state of the good performing ones. Therefore, the whole state of the model (including the weights) and the optimizer is saved in a checkpoint and transferred to another accelerator, where perturbations to the original hyperparameters are performed and the training is continued. While the training benefits from the AMP acceleration, the weights are still stored in full precision, as mentioned in Sec. 2. That is, too frequent checkpointing and weight transfers create an overhead that cancels out the performance gains. One solution is to increase the checkpointing interval. As shown in Fig. 2, saving the model weights in a checkpoint only every five instead of every epoch can reduce the overall runtime of the HPO process (compare red/green to orange/blue lines). This leads to a noticeable speed-up of the AMP training over the full precision training with a relative factor of ≈ 1.2.

The AMP heuristic of skipping the optimizer update if non-representable numbers are detected is already powerful. More targeted strategies, e.g., including some of the AMP parameters in the HPO search space did not improve performance. In comparison to one another, the ASHA algorithms seems to be the most favorable choice, achieving low runtimes with high test set performance.

Overall, the HPO runs that use AMP achieve slightly better performance in terms of accuracy and loss on the test set across almost all algorithms and datasets. This indicates that random noise introduced by the lower precision computations actually strengthens the generalization performance of models.

5 CONCLUSION AND FUTURE WORK

In this work, the speed-ups achieved by using different HPO algorithms (software) in combination with mixed precision capabilities of accelerators featuring Tensor Cores (hardware) have been investigated. It was shown that, depending on the checkpoint frequency or the model size, the computation can be accelerated by a factor of up to 1.56 on NVIDIA A100 GPUs. State-of-the-art accelerators, such as NVIDIA H100 GPUs, can run computations in even lower

Dataset	Metric	ASHA	BOHB	PBT	RAND
CFD	Runtime (AMP/full)	$\underline{3046\text{ s}}$ / 3992 s	$\underline{8994\text{ s}}$ / 13319 s	7428 s / $\underline{7383\text{ s}}$	$\underline{5253\text{ s}}$ / 6825 s
	Test mse (AMP/full)	$\underline{3.20 \times 10^{-3}}$ / 3.31×10^{-3}	$\underline{2.63 \times 10^{-3}}$ / 2.77×10^{-3}	$\underline{3.40 \times 10^{-3}}$ / 3.40×10^{-3}	$\underline{4.50 \times 10^{-3}}$ / 4.50×10^{-3}
cifar-10	Runtime (AMP/full)	$\underline{3293\text{ s}}$ / 3511 s	$\underline{9591\text{ s}}$ / 13858 s	2720 s / $\underline{2583\text{ s}}$	$\underline{3006\text{ s}}$ / 3394 s
	Test acc. (AMP/full)	$\underline{0.7798}$ / 0.7714	0.7768 / $\underline{0.7830}$	$\underline{0.6721}$ / 0.6093	$\underline{0.7677}$ / 0.7560
ImageNet-1k	Runtime (AMP/full)	$\underline{10844\text{ s}}$ / 16915 s	$\underline{26354\text{ s}}$ / 40495 s	5642 s / $\underline{5233\text{ s}}$	$\underline{6664\text{ s}}$ / 10249 s
	Test acc. (AMP/full)	$\underline{0.7006}$/0.6975	$\underline{0.7055}$ / 0.7042	$\underline{0.6021}$ / 0.5868	0.6758 / $\underline{0.6762}$

Table 1: Comparison of running different HPO algorithms with full precision and AMP training, averaged over three random seeds. Better results are underlined.

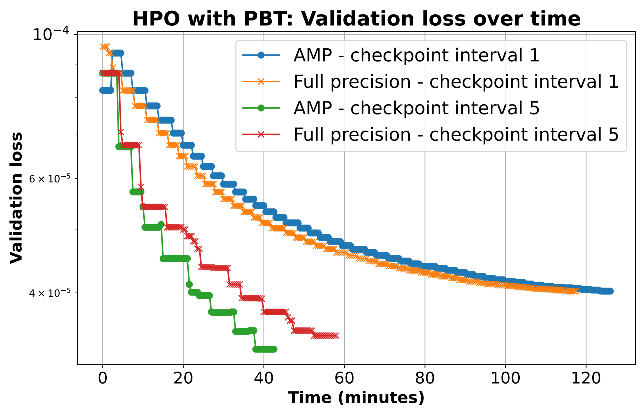

Figure 2: Comparison of running PBT using different checkpointing intervals on CFD dataset.

precision (primarily for Transformer models) with a potential to achieve even large speed-ups in the future. Additionally, other HPO frameworks that run on Message Passing Interface (MPI), e.g., DeepHyper [5] or Propulate [18], could be explored in addition to Ray Tune.

ACKNOWLEDGMENTS

The research leading to these results has been conducted in the CoE RAISE project, which receives funding from the European Union's Horizon 2020 – Research and Innovation Framework Programme H2020-INFRAEDI-2019-1 under grant agreement no. 951733. The authors gratefully acknowledge the computing time granted by the JARA Vergabegremium and provided on the JARA Partition part of the supercomputer JURECA at Forschungszentrum Jülich.

REFERENCES

[1] Marian Albers, Pascal S. Meysonnat, Daniel Fernex, Richard Semaan, Bernd R. Noack, Wolfgang Schröder, and Andreas Lintermann. 2023. Actuated Turbulent Boundary Layer Flows Dataset. https://doi.org/10.34730/5dbc8e35f21241d0889906136cf28d26
[2] Prasanna Balaprakash, Romain Egele, Misha Salim, Stefan Wild, Venkatram Vishwanath, Fangfang Xia, Tom Brettin, and Rick Stevens. 2019. Scalable Reinforcement-Learning-Based Neural Architecture Search for Cancer Deep Learning Research *(SC '19)*. ACM, New York, NY, USA. https://doi.org/10.1145/3295500.3356202
[3] Hadjer Benmeziane, Kaoutar El Maghraoui, Hamza Ouarnoughi, Smail Niar, Martin Wistuba, and Naigang Wang. 2021. A Comprehensive Survey on Hardware-Aware Neural Architecture Search. arXiv:2101.09336 [cs.LG]
[4] Yoojin Choi, Mostafa El-Khamy, and Jungwon Lee. 2020. Learning Sparse Low-Precision Neural Networks With Learnable Regularization. *IEEE Access* (2020). https://doi.org/10.1109/access.2020.2996936
[5] DeepHyper Development Team. 2018. "DeepHyper: A Python Package for Scalable Neural Architecture and Hyperparameter Search". https://github.com/deephyper/deephyper
[6] Xuanyi Dong, Lu Liu, Katarzyna Musial, and Bogdan Gabrys. 2021. NATS-Bench: Benchmarking NAS Algorithms for Architecture Topology and Size. *TPAMI* (2021), 1–1. https://doi.org/10.1109/tpami.2021.3054824
[7] Stefan Falkner, Aaron Klein, and Frank Hutter. 2018. BOHB: Robust and Efficient Hyperparameter Optimization at Scale. In *ICML*. 1436–1445.
[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *CVPR*. 770–778.
[9] Eray Inanc, Rakesh Sarma, Marcel Aach, and Andreas Lintermann. 2023. AI4HPC. https://doi.org/10.5281/ZENODO.7705421
[10] Max Jaderberg, Valentin Dalibard, Simon Osindero, Wojciech M. Czarnecki, Jeff Donahue, Ali Razavi, Oriol Vinyals, Tim Green, Iain Dunning, Karen Simonyan, Chrisantha Fernando, and Koray Kavukcuoglu. 2017. Population Based Training of Neural Networks. arXiv:1711.09846 [cs.LG] arXiv: 1711.09846.
[11] Kevin Jamieson and Ameet Talwalkar. 2016. Non-stochastic Best Arm Identification and Hyperparameter Optimization. In *AISTATS*. 240–248.
[12] Jülich Supercomputing Centre. 2021. JURECA: Data Centric and Booster Modules implementing the Modular Supercomputing Architecture at Jülich Supercomputing Centre. *JLSRF* 7 (2021), A182. http://dx.doi.org/10.17815/jlsrf-7-182
[13] Alex Krizhevsky. 2009. Learning Multiple Layers of Features from Tiny Images. (2009).
[14] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. 2017. Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization. *JMLR* 18, 1 (2017), 6765–6816.
[15] Liam Li, Kevin Jamieson, Afshin Rostamizadeh, Ekaterina Gonina, Jonathan Ben-tzur, Moritz Hardt, Benjamin Recht, and Ameet Talwalkar. 2020. A System for Massively Parallel Hyperparameter Tuning. In *MLSys*, Vol. 2. 230–246.
[16] Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, and Hao Wu. 2018. Mixed Precision Training. In *ICLR*. https://openreview.net/forum?id=r1gs9JgRZ
[17] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. 2015. ImageNet Large Scale Visual Recognition Challenge. *IJCV* 115, 3 (2015), 211–252. https://doi.org/10.1007/s11263-015-0816-y
[18] Oskar Taubert, Marie Weiel, Daniel Coquelin, Anis Farshian, Charlotte Debus, Alexander Schug, Achim Streit, and Markus Götz. 2023. Massively Parallel Genetic Optimization Through Asynchronous Propagation of Populations. In *LNCS*. 106–124. https://doi.org/10.1007/978-3-031-32041-5_6
[19] Eric Wulff, Maria Girone, and Joosep Pata. 2023. Hyperparameter optimization of data-driven AI models on HPC systems. *JPCS* 2438, 1 (2023), 012092. https://doi.org/10.1088/1742-6596/2438/1/012092