Exploring NCCL Tuning Strategies for Distributed Deep Learning

Majid Salimi Beni*, Ruben Laso†, Biagio Cosenza‡, Siegfried Benkner†, and Sascha Hunold*

*Faculty of Informatics, TU Wien, Austria
Emails: {majid.salimibeni, sascha.hunold}@tuwien.ac.at

†Faculty of Computer Science, University of Vienna, Austria
Emails: {ruben.laso.rodriguez, siegfried.benkner}@univie.ac.at

‡Department of Computer Science, University of Salerno, Italy
Email: bcosenza@unisa.it

***Abstract*—The communication overhead in distributed deep learning caused by the synchronization of model parameters across multiple devices can significantly impact training time. Although powerful GPU-GPU communication libraries, such as NCCL, are available, their default configurations have not been effectively adapted to varying hardware and workloads, which can result in lower performance.**

In this paper, we explore the tuning potential of NCCL and present an approach to tuning its parameters for distributed deep learning workloads. We identify efficient parameter configurations through an optimization process that explores the solution space defined by performance-related NCCL parameters. Experimental results on the Leonardo supercomputer, utilizing up to 64 GPUs, show significant performance improvements across micro-benchmarks and three deep learning models. For `ncclAllReduce` and `ncclAllGather`, we improved the bandwidth by factors of 112× and 36× in micro-benchmarks, respectively. The tuned NCCL parameter configurations reduced the training time of the models by up to 12.5%.

***Index Terms*—NCCL, Parameter Tuning, Collective Communications, Deep Learning, Multi-GPU**

I. Introduction

Deep Learning (DL) models require a significant amount of computational resources on multi-GPU clusters to be trained in a reasonable amount of time, even on cutting-edge GPUs. Current DL models with billions of parameters are too large to fit into the memory of a single GPU [1]. Therefore, distributed deep learning has become an essential approach, utilizing several GPUs and compute nodes to distribute the data, the model, or both. While distributed DL enables processing very large models, new challenges arise, as the communication overhead can severely impact training times [2], e.g., the time for model parameter synchronization can be as high as 90 % of the overall training time [3]. Collective Communication Libraries (CCLs) such as NVIDIA Collective Communications Library (NCCL) and ROCm Communication Collectives Library (RCCL) facilitate communication across multiple GPUs and nodes by providing efficient collective operations such as all-reduce and all-gather.

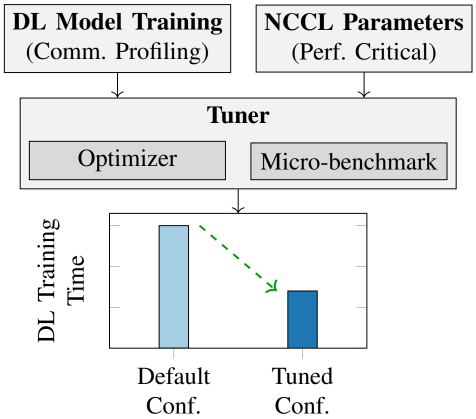


Fig. 1: Proposed strategy for tuning NCCL parameters and improving distributed deep learning performance.

CCLs provide several environmental variables that can be adjusted to enhance performance for specific workloads and hardware configurations. These parameters can be used, for instance, to select optimized collective algorithms (e.g., tree- or ring-based), communication protocols and channels, network transport, communication helper threads, and more, to improve hardware and topology awareness. Taking into account the underlying topology, NCCL employs built-in heuristics to select an effective combination of algorithms (`NCCL_ALGO`) and protocols (`NCCL_PROTO`). However, default parameter configurations of CCLs often fail to achieve the best performance on real-world hardware and workloads. Therefore, they should be tuned for hardware architecture, network, topology, and workload [4], [5].

In this paper, we explore the potential performance improvements that can be achieved by tuning NCCL parameters. As shown in Fig. 1, we first identify relevant message sizes and collective operations in DL models. Then, we filter performance-critical NCCL parameters and initiate an optimization process to obtain tuned configurations, which are later shown to reduce the training time of DL models.

Our contributions are: 1) An analysis of performance-critical NCCL parameters and their possible values. 2) A tuning strategy that combines an optimizer and a micro-benchmark to find tuned parameter configurations. 3) An experimental analysis of the proposed strategy and its impact on the training time of three DL models on 64 GPUs of the Leonardo supercomputer.

II. Related Work

The tuning of communication libraries in HPC has been an active research field. At the MPI (Message Passing Interface) level, several online and offline tuning strategies have been proposed [6], [7], [8]. Similarly, research on optimizing GPU-GPU communication has explored designing micro-benchmarks [9], developing new collective algorithms [10], [11], and selecting topology-aware collective algorithms for efficient GPU-GPU communication [4].

Other works [12], [13], [14] proposed different ways of synthesizing collective algorithms and generating topology- and hardware-aware collective algorithms that are tailored to the underlying hardware. Kim et al. [5] focused on PCIe-based GPU clusters and proposed a methodology that includes a profiling phase to identify the best paths for collective operations and to utilize the optimal paths through NCCL.

The works previously mentioned have explored different optimization potentials for CCL libraries, specifically focusing on the collective algorithm selection and design according to the available hardware. While selecting an effective algorithm for the collective operation and choosing an optimal communication path can significantly enhance performance, other NCCL parameters, such as network and thread configurations, are often overlooked and typically remain at their default settings. Although these parameters have been shown to enhance the performance of CCL libraries [15], the magnitude of their performance impact is still unexplored.

III. NCCL Tuning Methodology

We present our approach to explore the potential of tuning the NCCL environment variables to enhance the performance of distributed training of deep learning. To that end, we devise a tuning strategy composed of the following steps: filtering of NCCL parameters, profiling of deep learning models training, and offline tuning using a micro-benchmark.

A. Filtering of NCCL Parameters

According to the NCCL documentation (as of January 2025), 90 environment variables are exposed and can be used to control the behavior of the library. The first step in our strategy is to manually identify performance-related parameters that can be tuned to improve the bandwidth of the collective communication operations. As a result, the number of tunable parameters was reduced to 45. Additionally, we split the parameters according to their applicability to inter- and intra-node scenarios. Ideally, the number of tunable parameters could potentially be smaller to reduce the search space. However, we adopted a conservative approach to ensure that no important parameters and their interactions are overlooked.

B. Profiling the Training of Deep Learning Models

Since the training of various DL models may involve different collective operations and message sizes, we need to determine those with the biggest impact on runtime. Using NCCL's debugger, we profile 10 training epochs of each model to identify the most frequently used collective operations and message sizes.

C. Offline Tuning of NCCL Parameters

The last step in our strategy is to tune the NCCL parameters. Given the set of performance-related parameters, the target collective operation, and the message size, we use a Bayesian optimizer [16] to find a better configuration. These optimizers are known to be efficient in exploring the solution space and finding a better configuration in a limited number of iterations.

In the optimization process, different configurations are generated by the optimizer. Each configuration consists of a specific allotment of values of the performance-critical NCCL parameters. To determine the bandwidth of the collective operation, the set of NCCL parameters is forwarded to a micro-benchmark. This micro-benchmark runs that collective for a given number of iterations and computes the bus bandwidth according to NCCL's documentation. At the end of the optimization process, the best configuration is saved for later use in the model training.

IV. Experimental Results

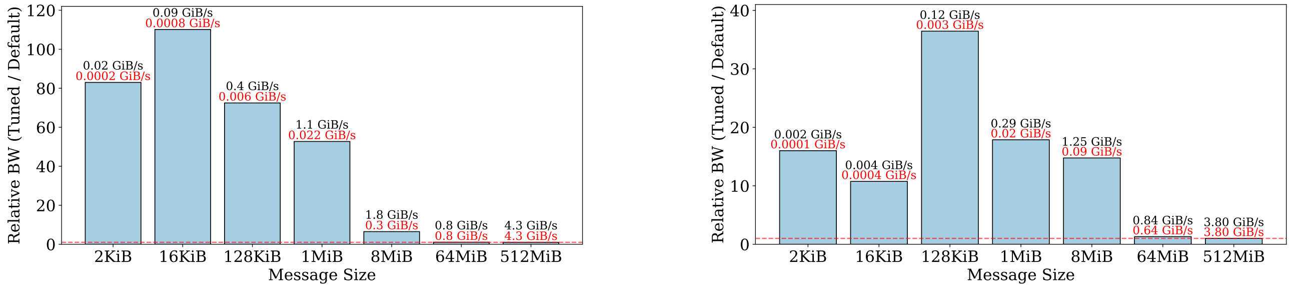
We carried out experiments on the Leonardo supercomputer in CINECA, Italy. Each node has 32-core Intel Ice Lake Xeon 8358 processors and four NVIDIA Ampere A100 GPUs. Intra-node communication uses NVLink 3.0, providing 200 Gbit/s between each pair of GPUs. Inter-node communication is provided by NVIDIA Infiniband HDR with a Dragonfly+ topology, organized into 23 groups, forming a 2-level fat tree, which provides a 100 Gbit/s bandwidth per port.

Related to software, we used Open MPI v4.1.4, CUDA 11.8, NCCL 2.19, Horovod 0.26.1 [17], and TensorFlow 2.10.0. For the Bayesian optimizer, we used Optuna v4.1.0 [18] with a Tree-Structured Parzen Estimator [16] sampler. The DL models used in the experiments are DenseNet121 [19], EfficientNetB0 [20], and NasNetMobile [21].

A. Impact of NCCL Parameter Tuning on Micro-Benchmarks

We applied our tuning strategy to a range of message sizes to evaluate the performance of the tuned and default configurations of the NCCL parameters for `ncclAllReduce` and `ncclAllGather`. For each message size and collective, we ran the optimizer for 30 minutes to explore the parameter space and find a better configuration. During that time, 200 to 300 configurations were evaluated by the micro-benchmark.

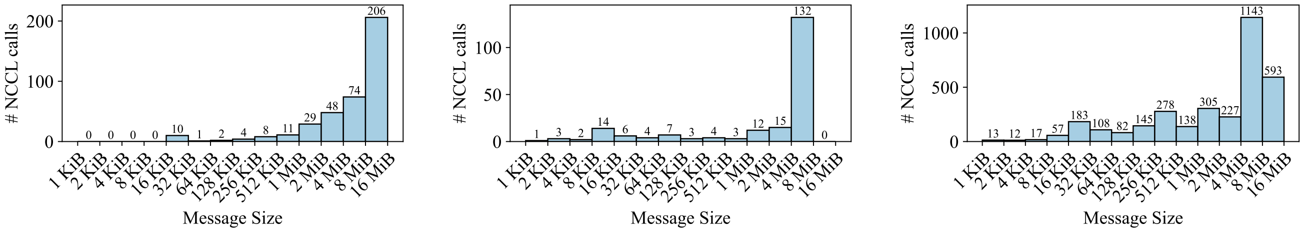
As shown in Fig. 2, tuning NCCL parameters has a high potential to improve performance for inter-node communication, especially for message sizes smaller than 64 MiB. For `ncclAllReduce` and message size of 16 KiB (cf. Figure 2a), the tuned bandwidth is $\frac{0.09\,\text{GiB/s}}{0.0008\,\text{GiB/s}} \approx 112\times$ higher than the one with the default configuration. For `ncclAllGather`,



(a) `ncclAllReduce`

(b) `ncclAllGather`

Fig. 2: Relative bandwidth in micro-benchmarks comparing tuned and default configurations. Raw bandwidths (GiB/s) are shown for the tuned (black) and default (red) configurations. Experiments on 16 nodes (64 GPUs). Higher is better.



(a) DenseNet121

(b) EfficientNetB0

(c) NasNetMobile

Fig. 3: Number of calls to `ncclAllReduce` after 10 training epochs as reported by NCCL for three deep learning models.

tuning the NCCL parameters led to a 36× higher bandwidth, in the best case (128 KiB, see Figure 2b). For larger messages, finding a better configuration than the default may require further exploration in the solution space, suggesting that NCCL is already well-tuned for these message sizes.

We also explored the tuning potential for intra-node communications on a single node of Leonardo (4 GPUs, not shown in the figures), achieving smaller improvements. In micro-benchmarks, the bandwidth was increased by 18 % for `ncclAllReduce` with 8 MiB, and by up to 7 % in the rest of scenarios. On a single node, the GPUs are fully connected via NVLink, and fewer parameters are involved, hence, there is limited room for significant performance gains.

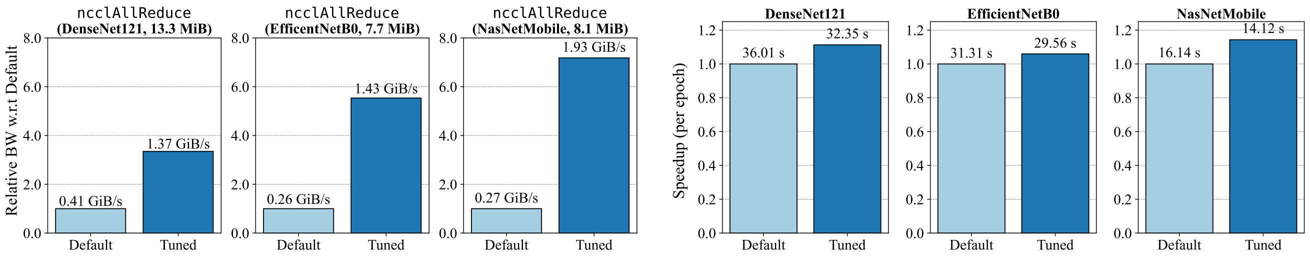
B. Impact of NCCL Parameter Tuning on Training DL Models

As mentioned in Section III, prior to the tuning, we should identify the most representative operations and message sizes for each model. Figure 3 shows the histogram of calls to NCCL collectives in the profiling phase. In all cases, `ncclAllReduce` is the most frequent operation, and the most relevant message sizes are 13.3 MiB, 7.7 MiB, and 8.1 MiB for DenseNet121, EfficientNetB0, and NasNetMobile, respectively. NasNetMobile requires the highest number of collective calls, 3301 in total, while DenseNet121 and EfficientNetB0 have only 393 and 206, respectively.

These message sizes serve as input for the optimization process that finds the tuned configurations. Figure 4 compares the default and tuned configurations in the micro-benchmarks and the speedup per training epoch. In micro-benchmarks (see Fig. 4a), we improved the bandwidth by 3.3×, 5.5×, and 7.2×, for the most relevant message sizes of DenseNet121, EfficientNetB0, and NasNetMobile, respectively. These improvements led to per-epoch training time reductions of 10.2 %, 5.6 %, and 12.5 %, respectively, as shown in Fig. 4b. The NasNetMobile improvement is bigger than for the other models primarily due to its higher communication intensity, with message sizes of approximately 8.1 MiB, which, as shown previously, have high tuning potential. DenseNet121 and EfficientNetB0 require longer training times per epoch than NasNetMobile, despite the smaller amount of communication, suggesting that their communication-to-computation ratio is lower and, consequently, their room for improvement.

V. Conclusions

In this paper, we explored the potential benefits of tuning NCCL parameters on distributed deep learning workloads. Using Bayesian optimization, we tuned NCCL's collective operations. For intra-node communications, we observed that bandwidth can be increased by up to 18 %. For inter-node communications, the gains were bigger, reaching up to 112× and 36× higher bandwidth for `ncclAllReduce` and `ncclAllGather`, respectively. Applying the tuned NCCL parameter configuration to the training of deep learning models, we reduced the per-epoch training time by up to 12.5 %. For future work, we plan to identify statistically significant parameters to narrow the search space and improve the optimization process by using heuristics to prioritize either the exploration or exploitation of key parameters.



(a) Bandwidth in micro-benchmarks.

(b) Speedup per training epoch in deep learning models.

Fig. 4: Comparison of the bandwidth in micro-benchmarks and speedup per training epoch for default and tuned configurations. Raw bandwidths (GiB/s) and training times (s) are shown over the bars. Experiments on 16 nodes (64 GPUs). Higher is better.

Acknowledgment

This work was partially supported by the Austrian Science Fund (FWF): project P 33884-N.

We acknowledge the CINECA award under the ISCRA initiative, for the availability of high-performance computing resources and support, and we acknowledge PRACE for awarding us access to Leonardo Booster hosted by CINECA, Italy.

References

[1] D. Narayanan, M. Shoeybi, J. Casper, P. LeGresley, M. Patwary, V. Korthikanti, D. Vainbrand, P. Kashinkunti, J. Bernauer, B. Catanzaro *et al.*, "Efficient large-scale language model training on GPU clusters using Megatron-LM," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC11)*, 2021, pp. 1–15.

[2] M. Khalilov, S. Di Girolamo, M. Chrapek, R. Nudelman, G. Bloch, and T. Hoefler, "Network-offloaded bandwidth-optimal Broadcast and Allgather for distributed AI," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC24)*. IEEE, 2024, pp. 1–17.

[3] D. Narayanan, A. Harlap, A. Phanishayee, V. Seshadri, N. R. Devanur, G. R. Ganger, P. B. Gibbons, and M. Zaharia, "PipeDream: Generalized pipeline parallelism for DNN training," in *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, 2019, pp. 1–15.

[4] G. Wang, S. Venkataraman, A. Phanishayee, N. Devanur, J. Thelin, and I. Stoica, "Blink: Fast and generic collectives for distributed ML," *Proceedings of Machine Learning and Systems*, vol. 2, pp. 172–186, 2020.

[5] H. Kim, J. Ryu, and J. Lee, "TCCL: Discovering better communication paths for PCIe GPU clusters," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, 2024, pp. 999–1015.

[6] M. S. Beni, B. Cosenza, and S. Hunold, "MPI collective algorithm selection in the presence of process arrival patterns," in *Proceedings of the 2024 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 2024, pp. 108–119.

[7] M. Salimi Beni, S. Hunold, and B. Cosenza, "Algorithm selection of MPI collectives considering system utilization," in *Proceedings of the Euro-Par Workshops*. Springer, 2023, pp. 302–307.

[8] S. Hunold and S. Steiner, "OMPICollTune: Autotuning MPI collectives by incremental online learning," in *Proceedings of the IEEE/ACM International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*. IEEE, 2022, pp. 123–128.

[9] M. Hidayetoglu, S. G. De Gonzalo, E. Slaughter, Y. Li, C. Zimmer, T. Bicer, B. Ren, W. Gropp, W.-M. Hwu, and A. Aiken, "CommBench: Micro-benchmarking hierarchical networks with multi-GPU, multi-NIC nodes," in *Proceedings of the 38th ACM International Conference on Supercomputing*, 2024, pp. 426–436.

[10] C.-H. Chu, P. Kousha, A. A. Awan, K. S. Khorassani, H. Subramoni, and D. K. Panda, "NV-Group: Link-efficient reduction for distributed deep learning on modern dense GPU systems," in *Proceedings of the 34th ACM International Conference on Supercomputing*, 2020, pp. 1–12.

[11] J. Huang, P. Majumder, S. Kim, A. Muzahid, K. H. Yum, and E. J. Kim, "Communication algorithm-architecture co-design for distributed deep learning," in *Proceedings of the 48th ACM/IEEE Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2021, pp. 181–194.

[12] A. Shah, V. Chidambaram, M. Cowan, S. Maleki, M. Musuvathi, T. Mytkowicz, J. Nelson, O. Saarikivi, and R. Singh, "TACCL: Guiding collective algorithm synthesis using communication sketches," in *Proceedings of the 20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, 2023, pp. 593–612.

[13] Z. Cai, Z. Liu, S. Maleki, M. Musuvathi, T. Mytkowicz, J. Nelson, and O. Saarikivi, "Synthesizing optimal collective algorithms," in *Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, 2021, pp. 62–75.

[14] W. Won, M. Elavazhagan, S. Srinivasan, S. Gupta, and T. Krishna, "TACOS: Topology-aware collective algorithm synthesizer for distributed machine learning," in *Proceedings of the 57th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2024, pp. 856–870.

[15] D. De Sensi, L. Pichetti, F. Vella, T. De Matteis, Z. Ren, L. Fusco, M. Turisini, D. Cesarini, K. Lust, A. Trivedi *et al.*, "Exploring GPU-to-GPU communication: Insights into supercomputer interconnects," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC24)*. IEEE, 2024, pp. 1–15.

[16] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," in *Proceedings of the 25th International Conference on Neural Information Processing Systems*, ser. NIPS'11. Red Hook, NY, USA: Curran Associates Inc., 2011, pp. 2546–2554.

[17] A. Sergeev and M. Del Balso, "Horovod: Fast and easy distributed deep learning in TensorFlow," *arXiv preprint arXiv:1802.05799*, 2018.

[18] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.

[19] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2017, pp. 4700–4708.

[20] M. Tan and Q. Le, "EfficientNet: Rethinking model scaling for convolutional neural networks," in *Proceedings of the International Conference on Machine Learning*. PMLR, 2019, pp. 6105–6114.

[21] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2018, pp. 8697–8710.