PCCL: Energy-efficient LLM Training with Power-aware Collective Communication

Ziyang Jia Laxmi N. Bhuyan
Department of Computer Science and Engineering
University of California, Riverside
Riverside, California
zjia016@ucr.edu, bhuyan@cs.ucr.edu

Daniel Wong
Department of Electrical and Computer Engineering
University of California, Riverside
Riverside, California
danwong@ucr.edu

2024 IEEE 42nd International Conference on Computer Design (ICCD) | 979-8-3503-8040-8/24/$31.00 ©2024 IEEE | DOI: 10.1109/ICCD63220.2024.00023

***Abstract*—The era of AI is witnessing a significant increase in energy consumption and carbon emissions from the execution of large language models (LLMs). Due to memory and compute requirements, it is necessary to distribute training and inference across many AI accelerators, such as GPUs. This paper focuses on distributed training that requires significant collective communication between accelerators; which often accounts for greater than half of training time. Besides LLMs, collective communication between GPUs is also common for many ML and HPC workloads.**

We first analyze the properties of collective communication operations in Nvidia Collective Communication Library (NCCL) and characterize the bandwidth, frequency, and energy properties of each collective communication operation. Then we propose PCCL, a Power-aware Collective Communication Library, based on NCCL, that can reduce power for communication kernels with dynamic voltage and frequency scaling (DVFS). PCCL identifies the optimal frequency for each collective communication call and precisely manages the GPU frequency accordingly in runtime. It can transparently lower the energy consumption of collective communication operations with negligible impact to throughput and performance. PCCL can reduce the energy of collective communication operations by ~27% and can reduce the end-to-end LLM training energy by 17.3%.

***Index Terms*—Frequency Scaling, DVFS, Collective Communication, LLM**

I. Introduction

The growth of Large Language Models (LLMs) in recent years has drawn considerable attention. As the LLMs grow in size and parameters, we require more computational resources, which results in greater power consumption and carbon emissions. Although ML inference is the majority consumer of energy usage [1], training still represents a significant proportion. Numerous research works have highlighted the large amount of energy consumed by deep learning and large language models [2] [3]. Therefore, to meet the needs of future LLM workloads and stay within the constraints of limited power budgets and carbon emissions, we need to optimize training and inference of LLM workloads to run more efficiently.

LLM training and inference require high-performance AI accelerators, such as GPUs, which consume a large amount of power [4] [5]. Due to the size of models and training data sets, it's common practice to distribute computation across multiple GPUs. As a result, collective communication between these GPUs becomes a critical aspect. Moreover, collective communication in LLMs consumes a significant amount of time, particularly as the number of GPUs scales up [6].

Given the considerable time consumed by communication, it also serves as a notable source of energy consumption. We found that the GPU tends to be very energy inefficient during collective communication operations. In addition, we identify that many collective communications can run at lower GPU frequencies without impacting the bandwidth.

To improve the energy efficiency of collective communication, we propose PCCL[1], a Power-aware Collective Communication Library, that employs Dynamic Voltage and Frequency Scaling (DVFS) techniques to improve the energy efficiency of collective communication while preserving its performance. The main contributions of this paper are:

- We perform a comprehensive workload and power characterization study of LLM training and observe that collective communication phases account for a significant amount of time during LLM training.
- Through micro-benchmark analysis, we identify that many collective primitives are insensitive to frequency by maintaining high bandwidth at significantly lower GPU frequencies and power consumption.
- We introduce PCCL, a Power-aware Collective Communication Library, that identifies the optimal frequency of a collective communication operation for achieving maximum power savings and limiting overheads.
- We evaluate PCCL and observe 27% energy savings of collective operations and 17.3% energy savings during end-to-end LLM training.

II. Background and Related Research

A. LLM workload and collective communication

Modern AI data centers are equipped with specialized AI accelerators to handle computationally intensive Large Model workloads. These large-scale systems comprise numerous compute nodes interconnected via high-speed networks, where each node is typically equipped with 4-8 powerful AI accelerator chips, e.g. GPU, TPU, FPGA, as depicted in Fig.1.

[1]PCCL is pronounced "Pickle", similar to how NCCL is pronounced "Nickle"

2576-6996/24/$31.00 ©2024 IEEE
DOI 10.1109/ICCD63220.2024.00023

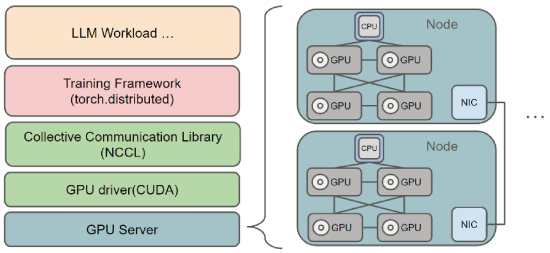

Fig. 1: Distributed ML System for LLM Workload

Collective communication libraries, such as Nvidia's NCCL implements multi-GPU communication primitives (e.g. AllReduce, AllGather) and point-to-point communication primitives (e.g. SendReceive). It is widely used as a communication backend for deep learning frameworks (e.g. Pytorch [7], TensorFlow [8]) for multi-GPU training. It facilitates transparent and high-performance data transfer between GPU devices, abstracting the complexities of inter-device communication.

B. Energy efficiency in LLM training systems

GPU DVFS: Dynamic voltage-frequency scaling (DVFS) is a widely adopted approach for achieving higher energy efficiency in computer systems. It can be applied to the whole data center [9]–[11], or specific hardware like CPU [12], GPU [13]–[15], DRAM [16] and SRAM [17]. All modern GPUs have the capability to dynamically adjust frequencies based on workload dynamics. For example, Nvidia GPUs uses GPU Boost, which automatically boosts and manages GPU frequency based on the available power and thermal headroom. However, these hardware-managed DVFS may not always be beneficial or optimal.

DVFS in ML training: Recent works have explored how DVFS can be applied to improve energy efficiency of ML training. For example, [18] characterized the impact of DVFS on end-to-end DNN training by statically assigning a DVFS state across training phases. Power-Inference accuracy Trading (PIT) [19] explores how precision trade-off and DVFS can be coordinated to save power in DNN workloads. EnvPipe [20] carefully applies DVFS to fine-grain tasks on non-critical paths in pipelined parallel ML training. Our work distinguishes from prior works by exploring how DVFS can be applied to the *communication* portion of ML training, not just the *computational* portion. We demonstrate that significant opportunities for DVFS exist in communication phases.

Collective Communication optimizations: There has been a notable number of works on the optimization of collective communication [21], [22]. These optimizations have focused on algorithmically improving the performance of collectives and their parameters in GPUs, such as NCCL, or collectives in CPUs through MPI collectives.

Prior works have also explored improving the energy efficiency of collective communication through DVFS in CPUs for distributed clusters over infiniband [23], [24]. However, these works target MPI collectives for HPC workloads, which has drastically different properties than modern ML workloads on GPUs [25].

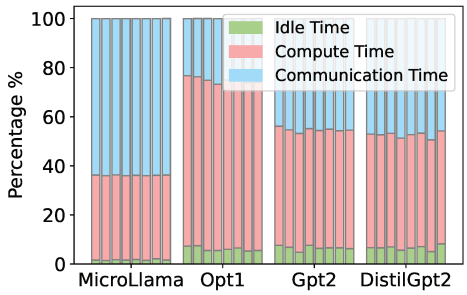

Fig. 2: Temporal breakdown of LLM training workloads.

III. Analysis of LLM Training on GPUs

In this section, we will address the challenges and opportunities in achieving energy-efficient collective communication in LLM training.

A. Evaluation Methodology

Our benchmarking and experiments were run on a real server with 8 Nvidia Telsa T4 GPUs, interconnected over PCIe. The server runs CentOS 7.9 with CUDA 12.3. Our LLMs run on PyTorch 2.3.0 with NCCL 2.18.0.

We use a variety of NCCL microbenchmarks (nccl-tests [26]) and LLM models to benchmark collective communication. We evaluate fine-tuning on various LLM models from HuggingFace [27] for end-to-end evaluation of LLM training. We evaluate MicroLlama (based on Llama2) with 300M parameters, OPT-1.5b with 1.5B parameters, Gpt2 with 124M parameters, and Distilgpt2 with 82M parameters. We utilize data parallelism to distribute training across 8 GPUs.

We use PyTorch Profiler [28] to collect traces of PyTorch activity and Holistic Trace Analysis (HTA) [29] to perform performance analysis of the traces. We use the NVIDIA Management Library (NVML) [30], which exposes APIs to read the GPU's on-board energy counters, to measure the energy and operating frequency of the GPUs. We created a profiling process that utilizes these NVML APIs to sample the energy and operating frequency at ~20ms intervals (due to NVML limitations).

B. Workload Characterization of LLM Training

In Figure 2, we present the breakdown of GPU kernels during training into the percentage of different categories for each model. Using Pytorch Profiler and HTA [29], we analyze the execution trace and classify the runtime activity into three categories: Compute, Non-compute, and Idle. The Compute category encompasses various computational kernels during LLM training. The Non-compute category encompasses communication-related kernels and memory-related kernels, typically collective communication through the Nvidia Collective Communication Library (NCCL) and memory copy between devices and hosts. However, we observed that the

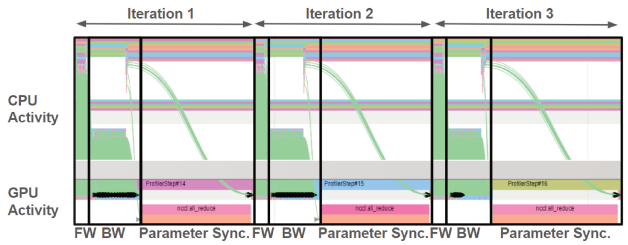

Fig. 3: PyTorch Profiler trace showing three iterations of MicroLlama training. Each iteration consist of a forward pass (FW), backward pass (BW), and parameter synchronization.

time of memory kernel in the model is less than 0.1%, so it can be neglected. Therefore, the Non-compute time reported can be interpreted as the time spent on communication operations during the training process. The Idle category encompasses time during training when neither Computation nor Communication kernels are running.

Observation: Communication time dominates LLM training. Figure 2 shows a stacked bar chart of the kernel breakdowns for all 8 GPUs for each workload. Across all workloads, communication time takes up the majority of LLM training time. For example, the communication time for MicroLlama, Opt1, Gpt2, and DistilGpt2 ranges from 23% to 64%. While it is widely known that LLMs require significant computational power for training, this result demonstrates that the distributed training nature of LLM training also incurs significant communication overheads. With larger models that require scaling out distributed training, it is expected that this communication overhead will only continue to grow. To this end, we argue that optimizations to communication should be as important as a focus on computation in LLM training.

Observation: Collective communication phases are longer than forward and backward passes. Figure 3 shows the phases of a typical training iteration. These traces were collected from PyTorch profiler and visualized through Chrome tracing. The figure shows 3 training iterations of MicroLlama, where each iteration consists of a forward pass (as FW), a backward pass (as BW), and a parameter synchronization stage (as Parameter Sync.). In data parallel ML training, the parameter synchronization stage is typically an all-reduce collective communication. As can be seen in this trace, collective communication often takes up two-thirds of a training iteration, significantly more than both forward and backward passes. In this trace, each iteration takes ~600ms. Within each iteration, the forward, backward, and parameter synchronization steps take ~40ms, ~110ms, and ~450ms, respectively.

C. Characterizing Energy Properties in LLM Training

In Figure 4, we show the energy properties of LLM training across 10 iterations. We sample and log both the power consumption and frequency of the GPU at ~20ms intervals. The collective communication phases are highlighted by the green regions, which were identified through timestamps from the Pytorch profiler and correlated with the timestamps from

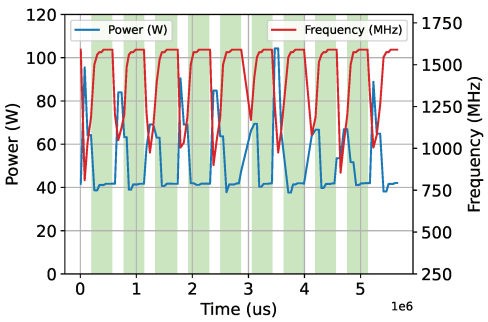

Fig. 4: Power and Frequency trace of MicroLlama training with NCCL. Communication phases are highlighted in green regions.

the power and frequency log. We present the results of a single GPU since all 8 GPUs exhibit similar power and frequency behavior, as shown in Figure 2.

Observation: Computation phases consume more power than communication phases. In Figure 4, we can see that each GPU consumes ~40W of power during the communication phase and 65-100W of power during the computation phase. Computational kernels and communication kernels have significantly different utilization of the GPU. Computational kernels are typically highly-optimized kernels from native Pytorch or vendor-specific libraries, such as cuDNN or cuBLAS. Due to this, computational kernels tend to have high utilization and use up all available hardware threads across all SMs. On the other hand, communication kernels from NCCL are highly optimized to improve the bandwidth and latency of communication. In our profiling, we observe that these kernels tend to utilize only a single SM and a single threadblock. These collective GPU kernels tend to perform limited computation (such as simple addition in reduce operations) and coordinate data copies through GPU-to-GPU peer access memory copy operations.

Observation: Counterintuitively, the GPU frequency is higher during communication phases than computation phases. Nvidia GPUs by default utilize a form of dynamic voltage frequency scaling (DVFS), called GPU Boost, to dynamically adjust the frequency of the GPU given the power and thermal headroom of the GPU. The Nvidia T4 GPU has a base frequency of 585MHz and a maximum boost frequency of 1590MHz. Therefore, when running workload on the GPU, the hardware will automatically modulate the frequency within this range to achieve maximum performance while satisfying the power and thermal limits of the GPU. Counterintuitively, we observe that the GPU operates at a higher frequency during communication phases and at a lower (but still boosted) frequency during computation phases. We believe that this counterintuitive behavior is due to the coarse-grain nature of GPU Performance States.

While the GPU is underutilized during communication phases, the GPU does not know to drop to a lower performance state to throttle the frequency down to save power. That is

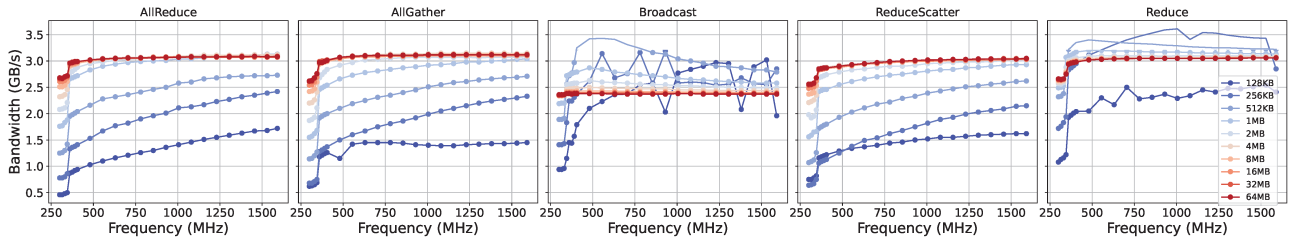

Fig. 5: Bandwidth of collective communications in NCCL. We vary the frequencies from 300MHz to 1590MHz (x-axis) for a range of data size from 128KB to 64MB.

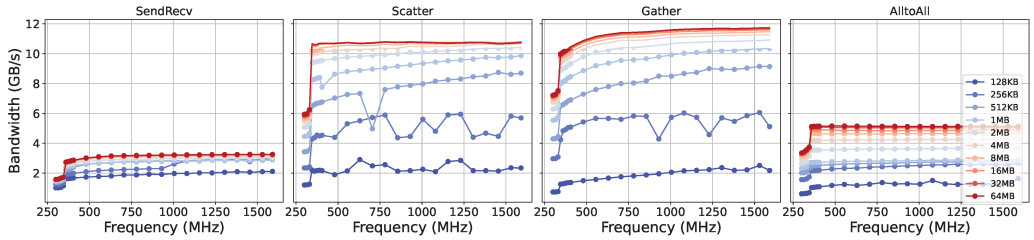

Fig. 6: Bandwidth of collective communications in NCCL. We vary the frequencies from 300MHz to 1590MHz (x-axis) for a range of data size from 128KB to 64MB.

because the Performance States (P-States) in GPUs tend to be granular and limited. Nvidia GPUs can potentially support 15 P-States, P0-P15, with P0 being the highest performance P-State. However, only certain P-States are actually implemented and used in practice. For example, we observe that our Nvidia T4 GPUs utilize only P0 when active and P8 when idle. Therefore, during communication phases, as long as a kernel is running, the GPU will be in the highest performance P0 P-State. The drop in frequency during computation is likely due to the GPU operating at a higher thermal output due to the higher utilization of the GPU. Similarly, during the communication phase when the GPU is under-utilized, the GPU has a lower thermal output, and thus, GPU Boost can operate at a higher boosted frequency.

IV. Analysis of Collective Communication

The aforementioned inefficiencies in hardware-managed DVFS presents an opportunity to improve the energy-efficiency during communication phases. NCCL supports five types of collective communication patterns: All Reduce, Broadcast, Reduce, Reduce Scatter, and All Gather. Additionally, it supports five types of point-to-point communication patterns: Send Receive, Scatter, Gather, and All to All [31]. For each communication pattern, we observe the impacts of GPU frequency on bandwidth by fixing SM frequency to a constant value using NVML while benchmarking with nccl-test [26].

The results are displayed in Figure 5 and Figure 6. Each figure shows the frequency (x-axis) vs bandwidth (y-axis) curve for a range of data sizes. The frequency range is from 300MHz to 1590MHz and the range of data size is from 128KB to 64MB.

Observation: In general, bandwidth increases as data size increases. For smaller data sizes, the communication time is shorter and does not saturate the bandwidth. As the data size increases, the amount of data communication necessary is sufficient to fully utilize the bandwidth until it reaches a peak bandwidth. Therefore, communication performance is better for larger collective operations. Two exceptions to this are Broadcast and Reduce, which show the highest bandwidth with data transfers around 256-512KB, before converging to a slightly lower bandwidth level at higher data transfer sizes. In general, for all workloads, once the data size exceeds 32MB, the frequency-bandwidth curve converges and stabilizes.

Observation: Bandwidth can be insensitive to GPU operating frequency. For all communication patterns, we observe that bandwidth remains relatively stable as frequency decreases. For example, in All Reduce, Figure 5, for data sizes above 2MB the bandwidth remains stable at 3GB/s even when the frequency is 450MHz At larger data sizes, the most frequency sensitive pattern is Send Receive, where bandwidth starts dropping slightly at 750MHz. Compared to the baseline scenario where collective communication phases run at 1590MHz (Figure 6), significant opportunity exists for lowering frequency for power savings.

Power savings opportunities are not just limited to large data sizes. For example, in All to All, the bandwidth is frequency insensitive for almost all data sizes. Additionally, for certain cases such as Broadcast and Reduce, bandwidth increases as frequency decreases. For example, Broadcast with 512KB data size can achieve bandwidth of 3.5GB/s at 500MHz, while only achieving 2.8GB/s at 1590MHz.

Observation: Collective communication data sizes in LLMs tend to be in frequency insensitive regions.

We observe that the LLM training exclusively uses All

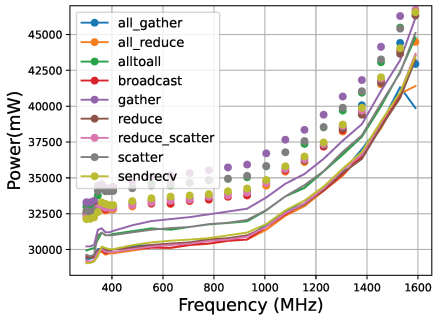

Fig. 7: Power consumption of collective operations as GPU frequency varies.

Reduce communication only. Other collectives, such as Broadcast, only occur during initialization of the model. We collected the message size of collective communication during training of various LLM models and observed that the collective data size tends to be either a few bytes or several hundred megabytes. We observe that for All Reduce, data sizes of a few bytes occur during initialization and data sizes of several hundred megabytes occur during training iterations.

For example, MicroLlama has All Reduce collective sizes of 62.5MB and 581MB. The vast majority of collectives are of tens of megabytes or hundreds of megabytes, these collectives already operate in a frequency-insensitive range, which provides significant opportunities to lower frequency for power savings with minimal bandwidth impact.

Opportunity: GPUs can save significant power at lower frequencies. Recall from Figure 4 that the baseline GPUs observe the maximum frequency during collective communication phases. We previously demonstrated that many collective communications can be frequency insensitive and maintain high bandwidth at significantly lower frequency. *This opportunity allows us to lower the frequency during collective communication phase with minimal performance impact.*

Figure 7 shows the frequency-power tradeoff curve for the various collective communication patterns. From Figure 5 and Figure 6, we observe that many collectives can lower frequency below 500MHz and still maintain bandwidth. From Figure 7, we can see that setting the GPU frequency to 500MHz for collective communication can cut the GPU's power consumption by ~13W (from 45W to 32W), or a power reduction of 29%. In the next section, we present our approach to harness these power savings opportunities.

V. PCCL: Power-aware Collective Communication

In this section, we introduce PCCL, a Power-aware Collective Communication Library. PCCL is built off NCCL and is a drop-in replacement for NCCL, requiring no code changes to ML frameworks such as PyTorch. PCCL enables collective communication operations to automatically adjust the frequency of the GPU to the lowest possible frequency while still maintaining bandwidth throughput.

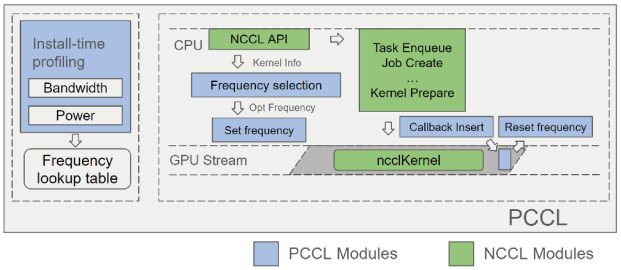

Fig. 8: Overview of PCCL. PCCL extends NCCL to enable DVFS management during collective communication phases.

A. Overview

Figure 8 shows an overview of PCCL. PCCL extends NCCL by adding a *Frequency selection* module that analyzes every collective call to identify the proper GPU frequency during the collective communication period. NCCL API calls are asynchronous and simply inserted kernel events that carry out the collective operation into the device stream. To set a custom frequency for a collective, PCCL injects a *frequency management event* on the host before the collective kernel event. PCCL also injects a callback function to reset frequency after the collective so that the GPU returns to the default GPU frequency for the future computation phase. To guide frequency selection, PCCL requires building a *frequency lookup table* and identifying an *overhead point* for setting frequency with NVML API calls. To build the frequency lookup table and determine the overhead point, PCCL requires a one-time profiling of the system, which can be done at library installation time or upon the first initialization call to PCCL.

B. System Profiling

PCCL requires a one-time profiling of the GPU's energy and bandwidth properties with respect to frequency and data size for various collectives. Since the GPU's energy and bandwidth relationship with frequency and data size does not change, this profile only needs to be done once per system. Many accelerated GPU libraries already take this approach by doing a one-time profiling run to tune various parameters of a library for a given hardware. For example, AMD's MIOpen library maintains a system and user performance database that profiles the hardware at installation time or on the first API call and stores various pre-tuned values for performance [32]. Therefore, this one-time system profiling does not incur any overhead in the critical path for LLM training or inference.

In PCCL, the profiling pass captures the bandwidth vs frequency vs data size curves for each collective, which are illustrated in Figure 5 and Figure 6. PCCL captures these curves by falling back to NCCL without any overheads from frequency selection and frequency management events.

Next, to account for the overhead of PCCL, PCCL profiles and captures a data size vs bandwidth curve for each collective where PCCL introduces the overhead of frequency selection and frequency management events with the frequency always being set to maximum frequency. The goal of obtaining this

Communication Operation	Overload Point
AllReduce	1MB
Broadcast	512KB
Reduce	512KB
AllGather	32MB
ReduceScatter	2MB
Scatter	64MB
Gather	64MB
AlltoAll	1MB
SendReceive	512KB

TABLE I: Overload point of communication operations. This represents the data size where the overhead of frequency management events no longer dominates.

profiling curve is to identify the overhead due to NVML-based frequency management APIs. In our experiments, the time to collect all of these profiling takes within 10 minutes.

C. Frequency Selection

During runtime, we need to determine *what* frequency a collective should run at, and *when* we should manage the frequency of a collective. To facilitate fast frequency selection at runtime, we analyze the profiled curves to create a *frequency lookup table* of minimum frequency given a collective and data size. In addition, each collective has a *management threshold* to guide when PCCL should manage frequency. To generate this frequency lookup table and guide frequency management decisions, we need to determine the following for each collective:

Minimum frequency point: This is the minimum frequency we can run without impacting bandwidth throughput.

Overhead point: This is the data size for a collective where the overhead of PCCL frequency management begins to dominate and impact bandwidth throughput.

To determine the *management threshold* of a collective, we take the overhead point. If the data size is greater than this threshold, we manage frequency as guided by the frequency lookup table. Otherwise, PCCL is unlikely to achieve energy savings or may have a large impact on bandwidth, so PCCL will leave frequency unmanaged.

1) Building Frequency Lookup Table: For each collective and data size, we need to determine the *minimum frequency* that results in similar a bandwidth to running at maximum frequency. To achieve this, we analyze each collective-data size curve and iterate from the maximum frequency to the lowest frequency and identify the point where the bandwidth begins to drop. The result of this analysis forms the *frequency lookup table*, where given a pair of collective and data size, we can determine the minimum frequency that we can run at without and bandwidth impact.

To minimize overheads, we limit the size of the frequency lookup table by only storing entries that are likely to be beneficial. If the minimum frequency of a collective-data point pair is determined to be the same as the maximum frequency, we do not store it in the lookup table. Therefore, the frequency lookup table should contain only entries that will likely lead to energy savings.

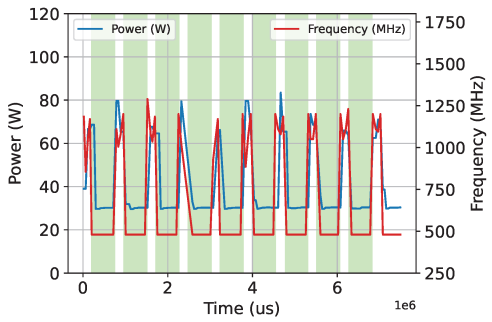

Fig. 9: Power and Frequency trace MicroLlama training with PCCL. PCCL achieves significantly lower frequency and power during communication phases.

Note that we cannot directly make management decisions based on this lookup table alone. Due to overheads of frequency management events in PCCL (mainly due to NVML APIs), we need to separately determine when to enable PCCL frequency management events.

2) Identifying Overhead Point: It is important for PCCL to determine whether we should manage frequency of a collective or not. Due to overheads of frequency management events, it is possible that the overheads of NVML can outweigh the benefits of lowering frequency, leading to no energy savings or lower bandwidth. Therefore, it is important for PCCL to identify this point. To obtain the overhead point, we compare the data size vs. bandwidth curves of NCCL with the data size vs. bandwidth curves of PCCL setting maximum frequency. The overhead point is the data size where these curves begin to diverge in achieved bandwidth. Table I shows the overhead points that were identified for all collectives.

During runtime, for every collective call, we compare the data size with this overhead point. If the data size is greater than this, we perform a lookup in the frequency lookup table to determine the frequency to run the collective at. If a minimum frequency is found, then we inject frequency management events to throttle the frequency. If there are no entries, then the collective is run at the default frequency.

D. Frequency Management Events

PCCL is implemented by extending NCCL. By maintaining the NCCL API, PCCL is a drop-in replacement for NCCL and requires no code changes to ML frameworks, such as PyTorch. All additional modules of PCCL (frequency selector and frequency management events) are entirely in NCCL. When a collective API is called, PCCL will first search the frequency selector to identify a frequency to run the collective at. We spawn a separate thread to perform frequency selection so it can be done in parallel with the initialization/setup tasks required by PCCL. The workflow of converting API calls to CUDA kernels which carry out the collective is unmodified from NCCL. Since collective API calls to PCCL/NCCL are non-blocking, they simply insert collective operations into device streams before returning.

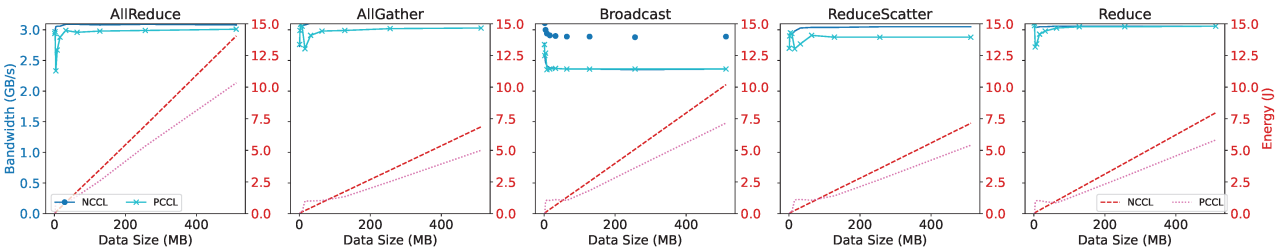

Fig. 10: Performance and energy comparison of 5 collective communications.

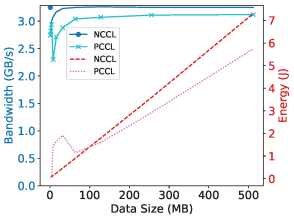

Fig. 11: Performance and energy comparison of Send Receive.

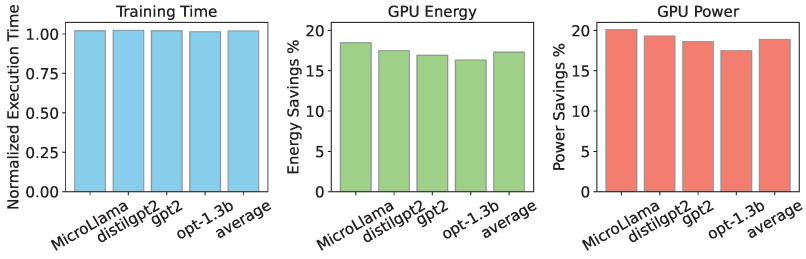

Fig. 12: End-to-end LLM training results. PCCL achieves significant energy and power savings with minimal performance overheads.

To ensure the frequency management events are timed correctly, we need to ensure that a frequency set event occurs before the collective task and a frequency reset event occurs after the collective task. For the frequency set event, we call this as soon as possible once we determine the frequency to run at. The frequency set event spawns a thread that uses`nvmlDeviceSetGpuLockedClocks()` to set the frequency of each GPU.

For the frequency reset event, we have to ensure that this occurs after the collective task. To achieve this, before the exit of the collective API, we insert a callback event into the device stream with `cudaLaunchHostFunc()`. Once the collective kernel completes, this callback event will trigger a callback function that spawns another thread to reset the frequency with `nvmlDeviceSetGpuLockedClocks()` for each GPU. Note, that the set and reset are performed with the same API, but with different parameters.

VI. Evaluation

In this section, we will evaluate the effectiveness of PCCL and compare it to the baseline state-of-the-practice NCCL collective communication library. The evaluation methodology was detailed previously in Section III-A. We will first evaluate PCCL on microbenchmarks and then perform end-to-end evaluation on LLM training.

A. Energy Consumption with PCCL

Figure 9 shows a power and frequency trace of PCCL while running the same workload as Figure 4. Here we can see that PCCL can successfully ramp down the frequency of every collective communication phase to 500MHz (from 1590MHz with NCCL). This results in collective communication phases to consume only ~33W of power (compared to ~45W with NCCL), *achieving ~27% power savings during collectives on average*.

Also note that during computation phases, the frequency is also more stable around 1000-1200MHz. Compared to Figure 4 with NCCL, the computational phases had a wider variance between 800-1200MHz. We speculate that the power savings during collective phases in PCCL lead to additional thermal headroom availability during the computation phases to enable more frequency boosting opportunities. Similarly, the power consumption was also more stable and lower during computation between 60-80W with PCCL, compared to 60-100W with NCCL.

B. Microbenchmark evaluation

We use nccl-test to evaluate the impact of PCCL on the bandwidth and energy across various collective data sizes. These results are presented in Figure 10 and Figure 11. Due to space limitations, we only show Send Receive for point-to-point communication as all other communications that use Send Receive as a primitive.

We can see that PCCL achieves bandwidth close to NCCL for almost all data sizes. For very small data sizes, PCCL experiences more bandwidth impact due to the overheads of setting/resetting frequency. These regressions can potentially be avoided by setting a more conservative management threshold to guide PCCL with small message sizes. For LLM training, this is not a major issue as most observed collectives (All Reduce) are in the order of hundreds of MBs, where PCCL performs well. In the region where LLMs most commonly operate, we also observe that PCCL typically achieves ~27% energy savings. For example, at 256MB data size for All

Reduce, NCCL consumes 6.9 Joules per collective operation, while PCCL consumes 5.3 Joules.

C. End-to-end LLM Training Evaluation

Execution Time: Figure 12 shows the results of evaluating PCCL on LLM training. PCCL achieves similar execution time compared to NCCL, with 1-2% slowdown at most.

Power: Despite this slowdown, PCCL saves significantly more power than NCCL due to aggressively throttling frequency during communication phases. In terms of power, PCCL observes an average of 18.9% power savings, with up to 20.1% with MicroLlama.

Energy: Recall from Figure 2 that these LLM models spend ~30%–60% of the time in communication. Despite that wide variation, the energy and power savings remain relatively similar across all LLM models. As we saw before, we also observe slight power savings during computation phases, even though PCCL does not explicitly manage those frequencies. Overall, PCCL achieves an average of 17.3% energy savings compared to NCCL.

VII. Conclusion

As the demand for LLM training grows, so does the resource and energy requirement. This work identifies that collective communication takes up a significant portion of LLM training and that GPUs tend to be highly energy-inefficient during collective communication phases. We showed that many collective communication operations are insensitive to GPU frequency and can run at lower frequencies without affecting bandwidth. We introduced PCCL, a Power-aware Collective Communication Library that efficiently lowers the frequency of collective communication phases. PCCL is a drop-in replacement for NCCL and does not require any program modifications. We demonstrate that PCCL can lower the energy of collective communication by 27% and end-to-end LLM training by 17%.

Acknowledgements

The research was partly supported by National Science Foundation under grants CCF-1907401, CNS-1955650, CNS-2047521, CCF-2324940, CCF-2324941 and CNS-2415202. We would also like to thank the anonymous reviewers for their invaluable comments and insights.

References

[1] D. Patterson, J. Gonzalez, Q. Le, C. Liang, L.-M. Munguia, D. Rothchild, D. So, M. Texier, and J. Dean, "Carbon emissions and large neural network training," *arXiv preprint arXiv:2104.10350*, 2021.

[2] E. Strubell, A. Ganesh, and A. McCallum, "Energy and policy considerations for deep learning in nlp," *arXiv preprint arXiv:1906.02243*, 2019.

[3] A. S. Luccioni, S. Viguier, and A.-L. Ligozat, "Estimating the carbon footprint of bloom, a 176b parameter language model," *arXiv preprint arXiv:2211.02001*, 2022.

[4] M. Hodak, M. Gorkovenko, and A. Dholakia, "Towards power efficiency in deep learning on data center hardware," in *Big Data*, 2019.

[5] A. Jahanshahi, H. Z. Sabzi, C. Lau, and D. Wong, "GPU-NEST: Characterizing energy efficiency of multi-GPU inference servers," *IEEE Computer Architecture Letters*, vol. 19, no. 2, pp. 139–142, 2020.

[6] H. Qi, L. Dai, W. Chen, Z. Jia, and X. Lu, "Performance characterization of large language models on high-speed interconnects," in *IEEE Symposium on High-Performance Interconnects (HOTI)*, 2023.

[7] S. Li, Y. Zhao, R. Varma, O. Salpekar, P. Noordhuis, T. Li, A. Paszke, J. Smith, B. Vaughan, P. Damania *et al.*, "Pytorch distributed: Experiences on accelerating data parallel training," *arXiv preprint arXiv:2006.15704*, 2020.

[8] M. Abadi and et. al., "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. [Online]. Available: https://www.tensorflow.org/

[9] S. Wang, Z. Qian, J. Yuan, and I. You, "A DVFS based energy-efficient tasks scheduling in a data center," *IEEE Access*, vol. 5, pp. 13 090–13 102, 2017.

[10] A. Jahanshahi, N. Yu, and D. Wong, "PowerMorph: QoS-aware server power reshaping for data center regulation service," *ACM Trans. Archit. Code Optim.*, vol. 19, no. 3, aug 2022.

[11] L. Zhou, C.-H. Chou, L. N. Bhuyan, K. K. Ramakrishnan, and D. Wong, "Joint server and network energy saving in data centers for latency-sensitive applications," in *IPDPS*, 2018.

[12] C.-H. Chou, L. N. Bhuyan, and D. Wong, "μDPM: Dynamic power management for the microsecond era," in *HPCA*, 2019.

[13] S. Bharadwaj, S. Das, K. Mazumdar, B. M. Beckmann, and S. Kosonocky, "Predict; don't react for enabling efficient fine-grain DVFS in GPUs," in *ASPLOS*, 2024.

[14] M. Chow and D. Wong, "CoFRIS: Coordinated frequency and resource scaling for GPU inference servers," in *IGSC*, 2024.

[15] A. Jahanshahi, M. Rezvani, and D. Wong, "WattWiser: Power & resource-efficient scheduling for multi-model multi-GPU inference servers," in *IGSC*, 2024.

[16] Q. Deng, D. Meisner, A. Bhattacharjee, T. F. Wenisch, and R. Bianchini, "Coscale: Coordinating cpu and memory system dvfs in server systems," in *MICRO*, 2012.

[17] J. Zhang and E. Sadredini, "Inhale: Enabling high-performance and energy-efficient in-SRAM cryptographic hash for IOT," in *ICCAD*, 2022.

[18] Z. Tang, Y. Wang, Q. Wang, and X. Chu, "The impact of GPU DVFS on the energy and performance of deep learning: An empirical study," in *e-Energy*, 2019.

[19] S. M. Nabavinejad, H. Hafez-Kolahi, and S. Reda, "Coordinated DVFS and precision control for deep neural networks," *IEEE Computer Architecture Letters*, vol. 18, no. 2, pp. 136–140, 2019.

[20] S. Choi, I. Koo, J. Ahn, M. Jeon, and Y. Kwon, "EnvPipe: Performance-preserving DNN training framework for saving energy," in *USENIX ATC*, 2023.

[21] U. Wickramasinghe and A. Lumsdaine, "A survey of methods for collective communication optimization and tuning," *arXiv preprint arXiv:1611.06334*, 2016.

[22] G. Wang, S. Venkataraman, A. Phanishayee, N. Devanur, J. Thelin, and I. Stoica, "Blink: Fast and generic collectives for distributed ML," in *MLSys*, 2020.

[23] L. Oden, B. Klenk, and H. Fröning, "Energy-efficient collective reduce and allreduce operations on distributed GPUs," in *CCGrid*, 2014.

[24] K. C. Kandalla, E. P. Mancini, S. Sur, and D. K. Panda, "Designing power-aware collective communication algorithms for infiniband clusters," *ICPP*, 2010.

[25] Z. Han, C. Gao, J. Liu, S. Q. Zhang *et al.*, "Parameter-efficient fine-tuning for large models: A comprehensive survey," *arXiv preprint arXiv:2403.14608*, 2024.

[26] NVIDIA, "Nccl tests," 2021. [Online]. Available: https://github.com/NVIDIA/nccl-tests

[27] Hugging Face, "Hugging face datasets," 2024. [Online]. Available: https://huggingface.co/datasets

[28] PyTorch, "Pytorch profiler," 2024. [Online]. Available: https://pytorch.org/tutorials/recipes/recipes/profiler_recipe.html

[29] Meta, "Holistic trace analysis," 2022. [Online]. Available: https://hta.readthedocs.io/en/latest/

[30] NVIDIA, "Nvidia management library (nvml)," 2024. [Online]. Available: https://developer.nvidia.com/management-library-nvml

[31] ——, "Collective operations." [Online]. Available: https://docs.nvidia.com/deeplearning/nccl/user-guide/docs/usage/collectives.html

[32] AMD, "Performance databse - miopen," 2024. [Online]. Available: https://rocm.docs.amd.com/projects/MIOpen/en/docs-5.0.0/perfdatabase.html