EnvPipe: Performance-preserving DNN Training Framework for Saving Energy

Sangjin Choi and Inhoe Koo, *KAIST;* Jeongseob Ahn, *Ajou University;* Myeongjae Jeon, *UNIST;* Youngjin Kwon, *KAIST*

https://www.usenix.org/conference/atc23/presentation/choi

This paper is included in the Proceedings of the 2023 USENIX Annual Technical Conference.

July 10–12, 2023 • Boston, MA, USA

978-1-939133-35-9

EnvPipe: Performance-preserving DNN Training Framework for Saving Energy

Sangjin Choi *KAIST*
Inhoe Koo *KAIST*
Jeongseob Ahn *Ajou University*
Myeongjae Jeon *UNIST*
Youngjin Kwon *KAIST*

Energy saving is a crucial mission for data center providers. Among many services, DNN training and inference are significant contributors to energy consumption. This work focuses on saving energy in multi-GPU DNN training. Typically, energy savings come at the cost of some degree of performance degradation. However, determining the acceptable level of performance degradation for a long-running training job can be difficult.

This work proposes EnvPipe, an energy-saving DNN training framework. EnvPipe aims to maximize energy saving while maintaining negligible performance slowdown. EnvPipe takes advantage of slack time created by bubbles in pipeline parallelism. It schedules pipeline units to place bubbles after pipeline units as frequently as possible and then stretches the execution time of pipeline units by lowering the SM frequency. During this process, EnvPipe does not modify hyperparameters or pipeline dependencies, preserving the original accuracy of the training task. It selectively lowers the SM frequency of pipeline units to avoid performance degradation. We implement EnvPipe as a library using PyTorch and demonstrate that it can save up to 25.2% energy in single-node training with 4 GPUs and 28.4% in multi-node training with 16 GPUs, while keeping performance degradation to less than 1%.

1 Introduction

Reducing carbon footprint is a worldwide mission. Experts estimate data centers take up 3% of the global carbon emission, which is roughly equal to the worldwide airline industry [2]. To mitigate carbon emissions, data center providers should actively explore energy-saving strategies for their operations. One significant area to address is the energy consumption associated with machine learning (ML) workloads, which constitutes a significant portion of overall energy usage. According to recent work, Google constantly spends 15% of its total energy running ML workloads for the past three years [22]. This study primarily focuses on energy saving in the context of multi-GPU deep neural network (DNN) training, which is a prevalent method employed in modern ML workloads.

There have been several approaches to save energy of GPU workloads. Common methods use GPU *Dynamic Voltage and Frequency Scaling* (DVFS), which seeks to identify the optimal frequency for the Streaming Multiprocessor (SM) clock or memory clock by balancing the tradeoff between performance and energy consumption [3, 7, 9, 12, 17, 27–29]. Recently, Zeus [29], considers the batch size and power limit to navigate the tradeoff between performance and energy consumption. It automatically finds the optimal configuration in recurring DNN training jobs based on the user-provided energy-efficiency importance. Although effective, these approaches leave several limitations.

First, they may have side effects by modifying user-provided hyperparameters. For example, Zeus adjusts the batch size of a training job which can potentially compromise statistical efficiency, even with optimally-tuned learning rates [24]. This issue becomes particularly challenging in non-recurring DNN training jobs where finding the batch size and learning rate pairs that maintain statistical efficiency is difficult. Second, it is difficult to determine how much performance degradation is acceptable at the cost of saving energy. Typically, the completion time of a training job varies and is unpredictable. Therefore, ML practitioners may not know how much delay they can accept. Furthermore, in a long-running training task, even a small performance degradation implies a significant delay. For instance, 10% degradation of a month-running task translates to three days. Third, these approaches primarily focus on individual GPU training jobs and do not adequately address the energy consumption associated with large-model training. Large model training utilizes multiple GPUs across multiple nodes with various parallelism techniques.

This work proposes EnvPipe[1], a new energy-saving DNN training framework. EnvPipe focuses on large model training using multiple GPUs with pipeline parallelism. EnvPipe addresses the limitation of previous approaches with the design goals: *No accuracy and performance degradation*. With the goals, users can run any DNN training jobs as if they run them without EnvPipe while saving energy under the hood. To preserve the original accuracy, EnvPipe does not modify any user-provided hyperparameters such as batch size and does not change data dependency while executing pipeline units. EnvPipe leverages the side-effect-free control knob only, SM frequency, to save energy. To avoid performance degradation, EnvPipe utilizes pipeline bubbles inevitably occurring when training large models with pipeline parallelism. EnvPipe selectively lowers SM frequency to reduce the energy consumption of pipeline units. This control stretches the execution time of pipeline units, but EnvPipe confines the degree of each stretch up to the available slack time of the bubbles, avoiding end-to-end performance degradation.

[1] **Env**elope + **Pipe**line Parallelism.

This design idea is generally applicable to any DNN training job where layer-wise partitioning is feasible, enabling training with pipeline parallelism across a large number of GPUs. However, realizing the design idea is challenging due to the following problems. **i)** To decide the value of SM frequency, EnvPipe needs to know the trend of clock speed and training performance. The trend varies according to GPU hardware, batch size, and the number of layers in a pipeline stage. Offline profiling to obtain this information is impractical. Instead, EnvPipe performs online profiling by sweeping SM frequencies to obtain the energy-saving curve under the given hyperparameters and GPU hardware. **ii)** How to schedule pipeline units decides the amount of bubbles that can be exploited. It is essential to ensure that a bubble exists after a pipeline unit of which execution time is stretched while ensuring that the next unit, which has data dependency, is sufficiently distant to avoid overall performance degradation.

EnvPipe is implemented as a library using the existing ML framework. The EnvPipe policy and mechanism are clearly separated, so developers can easily add required APIs to support a new ML framework. The current prototype of EnvPipe is implemented on the DeepSpeed [25] library and uses the existing GPU device driver to control SM frequency. We evaluate EnvPipe in real-world workloads: BERT, GPT, Megatron, and ResNet, and demonstrate the performance and energy saving on two GPU hardware: V100 and RTX3090. We perform evaluations of the workload in a single node (4 GPUs) and in multiple nodes (16 GPUs) and show that EnvPipe saves energy up to 25.2% and 28.4% in single-node and multi-node setups respectively while keeping performance degradation to less than 1%.

This paper makes the following contributions:

- We present the design of EnvPipe, a performance-preserving energy-saving DNN training framework that supports distributed training across multiple GPUs.
- EnvPipe preserves the original statistical efficiency by not modifying any user-provided hyperparameters and controls only the side-effect-free control knob.
- We demonstrate EnvPipe saves up to 25.2% and 28.4% energy saving in single- and multi-node GPU servers with less than 1% performance degradation.

The source code of EnvPipe is available on `https://github.com/casys-kaist/EnvPipe`.

2 Background

2.1 Large Model Training with Parallelism

Recent advancements in language models have focused on increasing the number of parameters, achieving impressive results on various challenging tasks such as language understanding, generation, and reasoning. Google's Pathways Language Model (PaLM), a 540 billion parameter model stacked up with numerous transformer decoder layers, has shown breakthrough results outperforming finetuned state-of-the-art models on various natural language tasks [8]. However, scaling up the model size comes with a cost of increased memory footprint, making it challenging to fit on a single GPU memory, even with the latest GPU like the NVIDIA H100 with 80GB. To efficiently train extremely large models, there have been several efforts to combine various parallelism techniques such as *data*, *tensor*, and *pipeline* parallelism [1, 11, 15, 16, 18–20, 30]

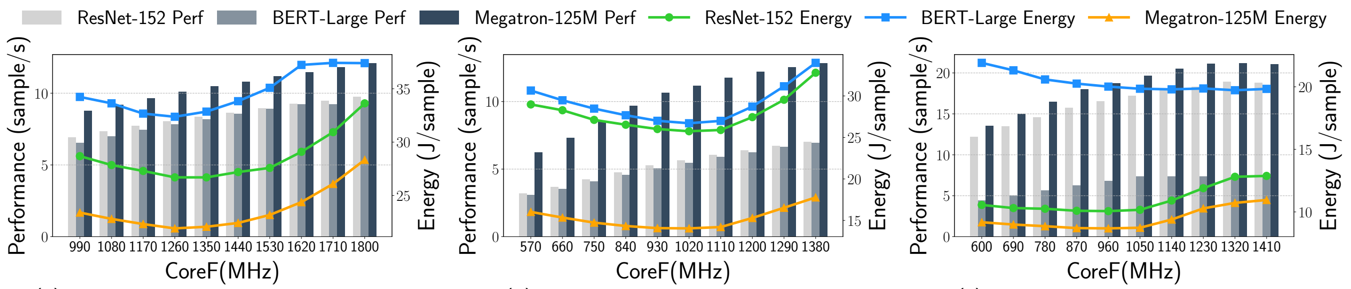
In this study, we focus on the pipeline parallelism [11, 15, 16, 18–20] which is a commonly used technique in training large DNNs whose models cannot fit on a single GPU. With pipeline parallelism, a model is vertically partitioned as evenly as possible to each worker (e.g., GPU) as pipeline stages. For transformer-based models such as GPT, each pipeline stage can have the same number of transformer decoder layers, balancing the execution time across the pipeline stages. To increase pipeline efficiency, the input batch is partitioned into multiple microbatches, and each worker handles the microbatches in a pipelined manner. There are two different approaches to synchronizing model parameters: synchronous and asynchronous. Synchronous pipeline parallelism (S-PP) ensures strict weight update semantics by periodic pipeline flushes. S-PP does not compromise the model's convergence but inevitably incurs pipeline bubbles which lower the training throughput [2]. Asynchronous pipeline parallelism (A-PP) relaxes weight update semantics and fully utilizes the pipeline throughput in a steady state by continuously pipelining microbatches without any pipeline flushes. A-PP hurts the statistical efficiency of the model and can fail to converge to the target accuracy [5].

In this work, we target S-PP which preserves the original statistical efficiency with strict weight update semantics. Due to pipeline flushes after every training iteration, pipeline bubbles are inevitable which lowers the pipeline throughput. Previous studies [15, 16, 20] focused on reducing the bubbles in S-PP. Rather than perceiving bubbles as an obstacle that slows down training, we consider pipeline bubbles as an opportunity to save energy in large model training.

2.2 Energy Scaling Valley Trend in GPUs

As it is convenient for ML practitioners to make use of GPUs rather than NPU-like accelerators for DNN workloads, modern cloud and data centers are operating a huge number of GPUs. However, when training DNN workloads, GPUs incur a significant fraction (e.g. about 70% according to [10]) of total power consumption in the whole system including other components such as CPU and DRAM. This high power consumption of GPUs during DNN training underscores the importance of optimizing their energy efficiency to reduce the overall energy consumption of cloud and data centers. To save energy in GPUs, previous studies [3, 7, 9, 12, 17, 27–29] have utilized *Dynamic Voltage and Frequency Scaling*

[2]GPU remains idle in pipeline bubbles



(a) Performance and energy of RTX3090 (b) Performance and energy of V100 (c) Performance and energy of A100

Figure 1: Energy scaling valley trend in modern GPUs

(DVFS) and focused on balancing performance and energy efficiency. DVFS is a widely studied technique in traditional CPUs to balance performance and energy consumption by scaling voltage/frequency in CPU cores. Generally, scaling down voltage/frequency saves energy but inevitably degrades performance, and GPUs tend to show a more complex energy scaling trend. Tang *et al.* [27] studied the energy scaling trend of various DNN training on modern datacenter GPUs and showed that energy saving is maximized on the middle-level core frequency and the energy consumption curve shows a valley trend when scaling core frequency.

We measure the training throughput and energy consumption by scaling the SM frequency in three different GPUs with various up-to-date DNN models in a single GPU training scenario. We use NVML [21], a library provided by NVIDIA, to adjust the SM frequency of GPUs. NVML can set the maximum limit of SM frequency and monitor the current energy consumption. Figure 1 exhibits that energy consumption decreases when lowering the frequency, but from the middle (e.g., 1350MHz and 1020MHz in RTX3090 and V100), this trend changes oppositely. This is because the training time is prolonged with lower SM frequency. Since energy consumption is related to both current power usage and overall execution time, if the end-to-end execution time increases at a faster rate than the rate of decrease in current power usage, the overall energy consumption increases. Thus, it is crucial to find the optimal point of SM frequency to achieve energy saving since the frequency cannot be lowered below the optimal point.

3 Energy-efficient DNN Training

This section describes the energy-saving problem of DNN training and discusses key insights that motivate the design of our system.

3.1 Objective and Constraints

DNN training is a complex and time-consuming process that places a significant emphasis on achieving statistical efficiency. Consequently, developing energy-efficient strategies for DNN training is a challenging task that can potentially lead to unintended side effects. In this section, we highlight several constraints that are crucial for ensuring the robustness of an energy-saving approach and mitigating any undesirable side effects.

No accuracy degradation. Given that training jobs are often already hyperparameter-searched, we do not modify any user-provided hyperparameters to ensure that the final accuracy after finishing the training is not compromised. Therefore, the way to achieve energy saving in this work is in sharp contrast to prior work that reduces energy consumption by changing hyperparameters, which can affect the final converged accuracy. For example, Zeus [29] studies how different batch sizes affect energy consumption when combined with a wide variety of GPU power scaling levels. The optimal combination chosen in Zeus thus alters depending on hardware and energy efficiency, which is the immediate consequence of the batch size in use. Optimizing energy consumption in this way is advantageous when users issue recurring DNN training jobs or can provide a set of batch sizes and corresponding hyperparameters that promise model convergence regardless of choice. Our target scenarios do not have that expectation from users. So, we decide to use control knobs that preserve the training's original statistical efficiency, such as controlling GPU SM frequency and dependency-aware pipeline scheduling.

No performance degradation. Given that curbing SM frequency to save energy affects DNN training speed, e.g., time taken to execute a single pipeline unit, we do not want to slow down the end-to-end training performance in exchange for energy savings for several reasons. First, from an ML practitioner's standpoint, it is difficult to determine how much performance degradation is acceptable across a wide range of training jobs. The completion time of a training job typically varies and remains unpredictable until the training is completed. Additionally, even a minor performance degradation in a long-running training task can result in a significant delay. For instance, a 10% slowdown in training time might seem small, but it can translate to a three-day increase in the duration of a month-long training job. Second, from a system administrator's standpoint, it is difficult to estimate an abrupt increase in GPU requests caused by prolonged DNN training jobs, which are already computationally expensive. Because GPU resources are highly contended and shared, prolonged DNN training jobs that occupy GPUs for extended periods contribute to increased GPU contention, necessitating the allocation of additional GPUs to alleviate resource bottlenecks.

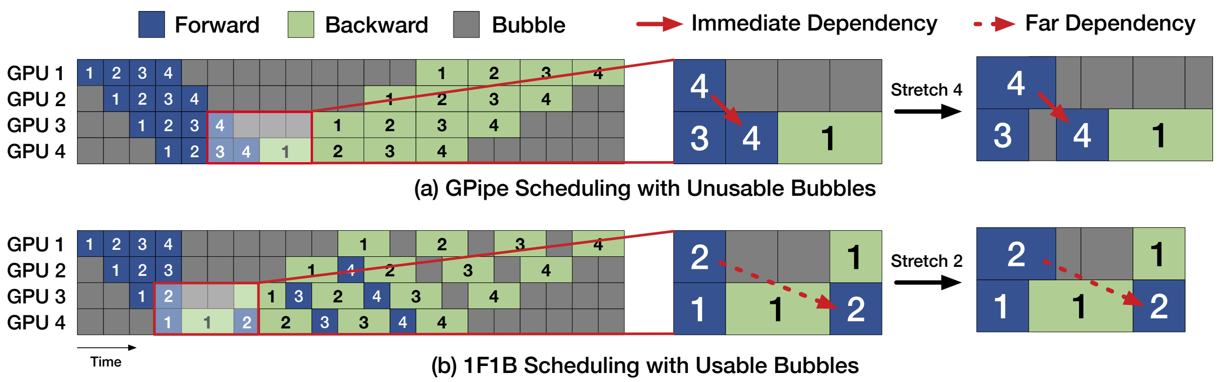


Figure 2: Comparison of two representative pipeline scheduling

Prior systems have centered on navigating the energy-performance tradeoff while keeping energy and time minimum, i.e., Pareto optimality. Unlike these approaches, our work primarily targets saving energy without sacrificing training time when training large models across multiple GPUs with pipeline parallelism, one of the most commonly used techniques in training large DNN models.

3.2 Insights

Our baseline model for distributed training, synchronous pipeline parallelism (S-PP), inevitably incurs bubbles, as shown in Figure 2. We selectively leverage these bubbles only when the pipeline units preceding them do not have immediate data dependencies. By reducing the SM frequency in these units, we can extend their execution time without necessarily delaying the start-up time of the subsequent units that rely on their outputs. This approach enables us to achieve energy savings while preserving performance. However, it is important to constrain the extent of stretching in each unit to a certain limit. This constraint ensures that no adverse performance delays occur as a result of elongated pipeline units.

Usable and unusable bubbles. Several S-PP designs have been proposed to schedule pipeline units during forward-backward computations of a single training iteration. Figure 2 shows the execution details of two representative S-PP designs, `GPIPE` and `1F1B`. The examples take four microbatches on four GPUs. In both cases, each microbatch execution goes through GPUs in order (GPU1 $\rightarrow$ GPU4) during the forward pass (FP) and then in reverse order (GPU4 $\rightarrow$ GPU1) during the backward pass (BP).

We observe that the performance-preserving energy-saving opportunity differs significantly in these two S-PP examples. To better understand this, we classify bubbles into two types: `Unusable` and `Usable`. A bubble is considered unusable when a stretched pipeline unit delays the overall training time. In Figure 2(a), stretching the forward pipeline unit of microbatch 4 (denoted as FP4) in GPU3 delays the start-up time of FP4 in GPU4 because of the `immediate dependency` caused by sending activation. This control delays the overall execution of the total pipeline. Even though plenty of bubbles are available after FP4 in GPU3, these bubbles are considered unusable, and exploiting unusable bubbles to save energy slows down training throughput, violating the constraints defined in § 3.1.

On the contrary, a bubble is considered usable when a stretched pipeline unit does not postpone the overall training time. For example, in Figure 2(b), stretching the forward pipeline unit of FP2 in GPU3 does not affect the execution of the backward pipeline unit of microbatch 1 (denoted as BP1) in GPU4 as these two units do not exhibit data dependency. FP2 in GPU4 exhibits the data dependency for activation communication, but it begins execution much later. We refer to this type of dependency as `far dependency`. Consequently, when utilizing the bubbles after FP2 in GPU3 for energy saving, none of the pipeline units in GPU4 gets penalized.

Based on this observation, we seek to exploit as many usable bubbles as possible for maximizing energy saving without performance degradation.

4 Design

4.1 Design Overview

This section presents the overview of our proposed system called EnvPipe. For a given DNN model and its hyperparameters, EnvPipe automatically tunes the order of pipeline units and generates an energy-saving plan controlling the SM frequency without any manual efforts from users. First, EnvPipe profiles the energy consumption of the DNN training job for each pipeline stage to understand performance and energy tradeoffs. Second, to increase the energy-saving opportunities, EnvPipe reschedules the pipeline units elaborately, increasing the amount of usable bubbles without breaking any data dependencies between the pipeline units. Last, EnvPipe finds an optimal SM frequency for pipeline units on the non-critical path to maximize energy savings without sacrificing training throughput.

Figure 3 depicts the overview of EnvPipe. EnvPipe consists of `online profiler` (§ 4.2), `scheduler` (§ 4.3), `frequency planner` (§ 4.4), and `execution engine`. At its

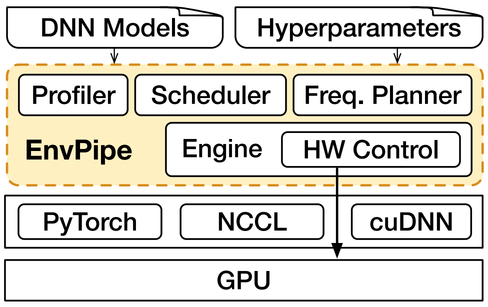


Figure 3: ENVPIPE Overview

core, ENVPIPE design clearly separates policy from mechanism. ENVPIPE generates a performance-preserving energy-saving plan by executing in the workflow of online profiler $\rightarrow$ scheduler $\rightarrow$ frequency planner (policy). The execution engine runs the energy-saving plan (mechanism).

Policy: Building the energy-saving plan. At first, ENVPIPE runs the `online profiler` to construct the energy valley curves for each pipeline stage when training the given DNN model. Based on the online profiling result, ENVPIPE `scheduler` decides the best schedule of forward and backward execution units, which creates plenty of usable bubbles. Using the scheduling decision, ENVPIPE runs the `frequency planner`. It lowers the SM frequency of all pipeline units inside the outermost path of the total pipeline (we call it *envelope*) to the optimal value identified from the energy valley curve. At this point, it is likely to degrade the training performance because execution units inside the envelope are stretched. To avoid performance degradation, ENVPIPE identifies the performance-critical path and reconfigures the SM frequency (i.e., undo lowering SM frequency) of all units in the performance-critical path to avoid performance slowdown.

After these steps are completed, ENVPIPE obtains the energy-saving plan that specifies I) a schedule (placement) of forward and backward pipeline units, and II) SM frequency value of each pipeline unit, which achieves energy saving without degrading performance.

Mechanism: Executing the energy-saving plan. The execution engine provides APIs for the `online profiler` and the `frequency planner`. Internal APIs used in the ENVPIPE's execution engine are translated to ML platform-specific APIs (e.g., PyTorch API calls). The engine includes HW control APIs communicating GPU device driver to control SM frequency.

ENVPIPE is implemented as a user-level library to invoke APIs of underlying ML platforms, providing an easy-to-use, platform-independent way to control multi-GPU pipeline scheduling and energy consumption. In addition, due to this clean separation of policy and mechanism, ENVPIPE can be applicable to any ML platform by implementing required APIs in the execution engine to support the ML platform [3].

[3]The current implementation supports PyTorch only

4.2 Fine-grained Online Profiling

As shown in Figure 1, energy consumption shows the valley trend according to SM frequency. The form of valley curves depends on GPU hardware, batch size, model architecture, and the method of splitting the model for pipelining. Therefore, ENVPIPE runs the *online* profiler to obtain the energy valley curve from given DNN models, GPU hardware, and hyperparameters. For each pipeline stage, the online profiler sweeps available ranges of SM frequency and measures energy consumption to find the optimal SM frequency that maximizes energy saving. The profiling steps are seamlessly integrated into the training procedure, allowing ENVPIPE to continue training with the weight version obtained from the profiling steps. The energy valley curves for each pipeline stage are generated within 100 steps and just 5 steps per frequency are enough to detect the optimal point where the energy-saving trend changes oppositely. Since training a model usually requires thousands to millions of steps, the overhead of the online profiler can be considered negligible. Note that the SM frequency of a pipeline unit cannot be lowered below the optimal energy-saving point which stretches the execution time of the pipeline unit to about 20 - 25% in our GPU settings. In addition, the online profiler measures the execution time at maximum and optimal SM frequency's forward and backward pass, and available GPU memory, which is used in the scheduling phase.

4.3 Scheduler: Utilizing Bubble

Design problems. As discussed in § 3.2, the scheduling of forward and backward pipeline units determines the amount of usable bubbles. When making scheduling decisions, it is important to consider two key questions: 1) how to effectively identify usable and unusable bubbles, and 2) how to optimize the utilization of usable bubbles by scheduling pipeline units.

Identifying usable bubbles. To answer the first question, we first need to identify pipeline units in the performance-critical and non-critical paths. Figure 4(a) shows bubbles and the performance-critical path (red boxes). It is important to note that usable bubbles are placed after pipeline units of the non-performance-critical paths. On the contrary, bubbles after the pipeline units of performance-critical paths are unusable. For example, if we stretch BP8 in GPU3 (★) which is on the performance-critical path to use the following bubbles, it will delay the start of BP8 in GPU2 (♠) causing performance degradation in the overall pipeline execution.

Optimizing utilization of usable bubbles. To answer the second question, we should consider the stretch limit of the pipeline units. Recall that there is a limit for the pipeline unit to get stretched because ENVPIPE does not set SM frequency below the optimal point, which is usually about 20 – 25% (§ 4.2). To optimize the utilization of usable bubbles, ENVPIPE should distribute the usable bubbles since a certain group of bubbles at the front of the pipeline with a long idle

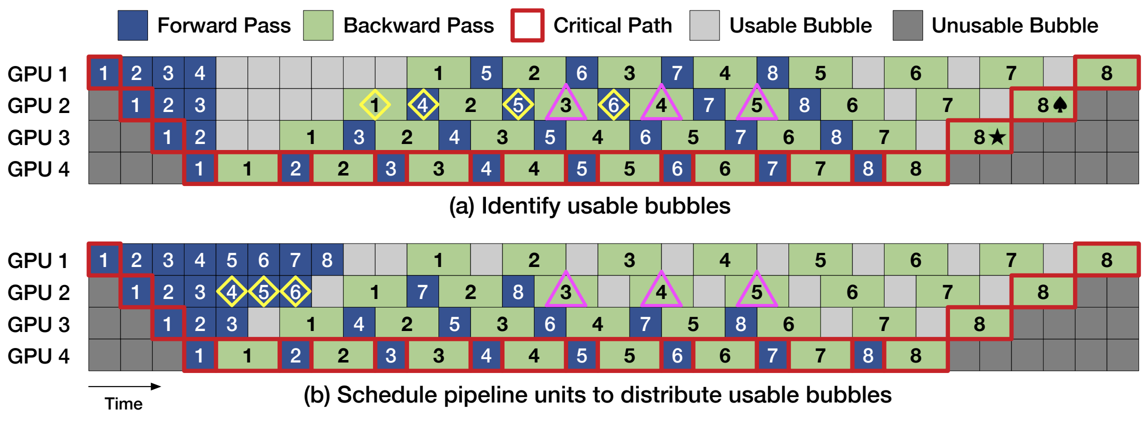


Figure 4: EnvPipe scheduling

time can be underutilized. For pipeline units to utilize the usable bubbles at their best, the usable bubble must be evenly distributed throughout the pipeline execution.

The EnvPipe scheduler operates in two phases: initialization and rescheduling. During initialization, EnvPipe employs a proven method for scheduling pipeline execution, which has been successfully deployed in a wide range of environments. Next, in order to enhance the utilization of usable bubbles, EnvPipe scheduler reschedules pipeline units in a manner that evenly distributes these bubbles while preserving the original data dependency between pipeline units.

Initialization. Among various existing approaches, we select one of the known methods that have the fewest units on the performance-critical path because it contains more usable bubbles. As observed in § 3.2, for the 1F1B schedule, pipeline units only along the outermost path (denoted as *envelope*) are on the performance-critical path. Therefore, we select the 1F1B schedule, which has the minimum number of pipeline units on the performance-critical path, as a starting point and further reschedules the pipeline units based on this initialization.

Rescheduling pipeline units. After initialization, EnvPipe reschedules pipeline units in order to reserve usable bubbles right behind pipeline units and to distribute usable bubbles among pipeline units. To save energy consumption, EnvPipe can stretch pipeline units up to the slack time made by the following bubble. Figure 4(b) shows the result of scheduling Figure 4(a). EnvPipe moves FP units to upfront usable bubbles (e.g., FP4, FP5, and FP6 in GPU2 ◇), generating usable bubbles after backward units (e.g., BP3, BP4, and BP5 in GPU2 △).

EnvPipe can compute how many FPs can be rescheduled and stretched by computing the available slack time of bubbles. When rescheduling the FP units, EnvPipe considers the following conditions. I) EnvPipe never breaks the data dependency for sending and receiving activations and gradients. For instance, FP3 in GPU3 starts only after the activation is sent from FP3 in GPU2. This is essential for preserving the original data dependency of the pipeline execution. II) Because EnvPipe moves forward units upfront, it needs to hold additional activation generated by each forward unit, which uses extra GPU memory. The memory used by activation is freed after the corresponding backward unit consumes it. The size of memory used by each activation is obtained by the online profiler. Therefore, EnvPipe computes available memory to hold the activations and only reschedules a certain number of forward units that can fit in the available memory to avoid an out-of-memory error.

Algorithm 1 Frequency Planner

```
1: while True do
2:     $ExecutePipelineStep()$
3:     $criticalPath \leftarrow FindCriticalPath()$
4:     if $criticalPath \neq$ outer envelope of total pipeline then
5:         $ReconfigureCriticalPath(criticalPath)$
6:     else
7:         break
8:     end if
9: end while
```

Figure 5: Frequency Planner Algorithm

4.4 Frequency Planner: Minimizing Performance Impact

Design problems. According to the scheduling decision, the goal of the frequency planner is maximizing energy saving while minimizing performance impact (less than 1%) by controlling SM frequency. To achieve this, the system leaves the SM frequency at its maximum for units on the performance-critical path and lowers the frequency to its energy-optimal value (obtained from the online profiler) for units not on this path using the available slack time of usable bubbles. However, this does not guarantee minimal performance degradation, since not all bubbles can accommodate the stretched pipeline units inside the envelope. So the question is how to selectively reconfigure the SM frequency to minimize performance impact. In addition, when reconfiguring the frequency of units on the performance-critical path, it is possible that the path may change. To prevent performance slowdowns, the system must be able to efficiently identify the new performance-

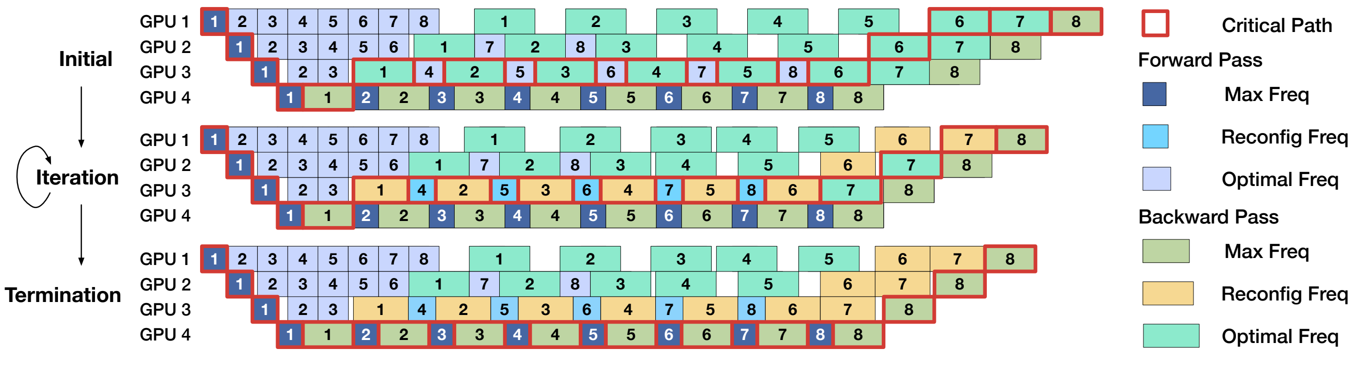
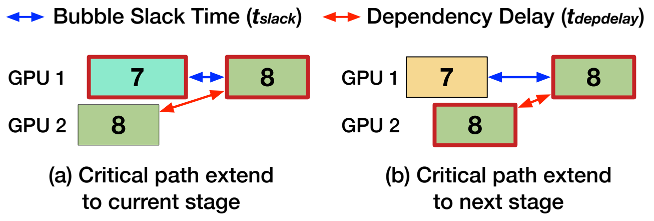


Figure 6: ENVPIPE frequency planning

critical path. This leads to the second design question: how can the performance-critical path be identified quickly and effectively?

Our strategy. Initially, ENVPIPE sets the energy-optimal SM frequency to pipeline units on non-critical paths which is all pipeline units inside the outer envelope of the total pipeline. To achieve the design goal, ENVPIPE runs an iterative algorithm as shown in Figure 5. Figure 6 illustrate the steps. After executing a single pipeline step, at Line 3, the algorithm finds the critical path of the executed pipeline (**Initial** of Figure 6). Then, at Line 5, it reconfigures SM frequency of pipeline units on the detected performance-critical path to avoid performance slowdown (**Iteration** of Figure 6). It repeats the *find and reconfigure* steps until the performance-critical path becomes the outer envelope of the pipeline. The algorithm stops when the critical path is identical to the outer envelope (**Termination** of Figure 6). The termination condition of the iterative algorithm ensures that the performance-critical path is the same as that of running pipeline parallelism without using ENVPIPE while our frequency planner stretched as many pipeline units as possible on the non-critical path.

Algorithm for finding the critical path. Figure 7 shows the algorithm to identify the performance-critical path. The algorithm incrementally builds the critical path backward. *current* in Figure 7 is a cursor, pointing to pipeline units, and moves backward. Initially, the cursor starts with the last unit (BP8) in GPU1. The algorithm repeatedly updates the cursor to point to the next pipeline unit to insert into the critical path until the cursor reaches the starting unit (FP1) in GPU1. At line 9, the algorithm decides the next pipeline unit to extend the critical path backward. Figure 7(a) and (b) illustrates the idea. Recall that the critical path consists of pipeline units that delay the overall execution time when stretched. Thus, we should find the pipeline unit that is affecting the start time of the pipeline unit that the cursor is currently pointing at. Let's assume the current cursor points to BP8 in GPU1. The algorithm finds the pipeline unit that is affecting the start time of BP8 in GPU1 by comparing the slack time — the bubble between BP7 and BP8 in GPU1 (t_{slack} of *current*) — and the dependency delay — spare time between the end of BP8 in GPU2 and the start of BP8 in GPU1 ($t_{depdelay}$). If t_{slack} of *current* $< t_{depdelay}$ (Figure 7(a)), the precedent pipeline unit of the same GPU is affecting the start time of BP8. So the algorithm updates the cursor to BP7 in GPU1 to add to the critical path. Otherwise, the pipeline unit that has the data dependency from the next GPU is affecting the start of BP8. Thus, the cursor is updated to BP8 in GPU2 and is added to the critical path. After that, the algorithm repeats the same step iteratively. The algorithm ends when the cursor reaches the first forward unit in GPU1, which is the first pipeline unit on the critical path that decides the starting time of the overall pipeline execution step.



Bubble Slack Time (t_{slack})
Dependency Delay ($t_{depdelay}$)
GPU 1
GPU 2
7
8
8
(a) Critical path extend to current stage
(b) Critical path extend to next stage

Algorithm 2 Finding critical path

```
1: $GPUID$: e.g., GPU 1 - GPU 4
2: $t_{slack}$: Slack time of the precedent bubble
3: $t_{depdelay}$: Delay between $pipelineUnits$ with data dependency
4: $current \leftarrow$ last backward $pipelineUnit$ in GPU 1
5: $criticalPath$.insert($current$)
6: while $current \neq$ first forward $pipelineUnit$ in GPU 1 do
7:     $n \leftarrow GPUID$ of $current$
8:     $k \leftarrow microbatchID$ of $current$
9:     if $t_{slack}$ of $current < t_{depdelay}$ then
10:        $current \leftarrow$ the previous $pipelineUnit$ in GPU $n$
11:    else
12:        if $current$ is forward $pipelineUnit$ then
13:            $current \leftarrow pipelineUnit$ $k$ in GPU $(n-1)$
14:        else
15:            $current \leftarrow pipelineUnit$ $k$ in GPU $(n+1)$
16:        end if
17:    end if
18:    $criticalPath$.insert($current$)
19: end while
```

Figure 7: Finding critical path

Algorithm for reconfiguring critical path. EnvPipe reconfigures the pipeline units on the critical path by increasing a small amount of SM frequency. The reconfiguration should be done in small steps in an iterative way for two reasons. First, by increasing the frequency in small increments, the stretched execution time can be gradually shortened, likely finding the point of fully utilizing the slack time of the usable bubbles. Second, due to the complex data dependencies between pipeline units, the critical path may change after reconfiguring pipeline units on the critical path. By reconfiguring in small steps, it will be less likely to unnecessarily increase the frequency of pipeline units on the non-critical path, since the critical path may have changed during the reconfiguration process.

The key question in the algorithm is identifying which pipeline units on the critical path should be reconfigured. To determine this, we use the observation of the trend in the energy-scaling valley curve shown in Figure 1. From the optimal point (minimum energy), as the SM frequency increases, the energy consumption and performance increase at different rates. Therefore, EnvPipe defines the *performance-energy utility* as the ratio of performance increase to energy increase for a frequency increase. In general, the performance-energy utility diminishes as it is further from the optimal point. Thus, to maximize the utility, the system should prioritize reconfiguration of SM frequency close to the optimal point.

Using this observation, EnvPipe takes a `balanced` approach. After finding the critical path, EnvPipe finds the pipeline units with the minimum SM frequency (i.e., closest one to the optimal value) and increases their SM frequency. This allows for the SM frequencies of all pipeline units on the critical path to be balanced as much as possible. For comparison, the system also implemented a simple approach called `greedy`. This approach selects pipeline units backwards from the end of the critical path and reconfigures them until their frequency is the maximum default frequency of a GPU. This approach also achieves the performance goal by increasing the SM frequency of pipeline units on the performance-critical path, but not in a balanced way.

4.5 Discussions

4.5.1 Size of Bubble and Energy Saving

The size of the bubble highly influences the achieved energy saving. The opportunity to leverage pipeline bubbles increases as the size of the bubble increases thus leading to higher energy savings. Since EnvPipe does not change the user-provided hyperparameters, achieved energy saving may differ according to the user-provided input or bubble-reducing methods that were studied in previous S-PP works [15, 16, 20].

Number of microbatches. The size of the bubble gets amortized over the number of micro-steps. Thus, as the number of microbatches increases, the fraction of the pipeline bubble decreases. For the portion of the bubble to be minimized, the number of microbatches should be larger than the number of pipeline stages. However, increasing the number of microbatches indefinitely is not possible because increasing the number of microbatches leads to an increase in global batch size. Even with a carefully tuned learning rate, there is a maximum limit in global batch size to preserve the statistical efficiency [24]. We show as a sensitivity study how the number of microbatches affects energy saving.

Partition method of pipeline stages. The partition method to split pipeline stages affects the size of pipeline bubbles. Pipeline bubbles are minimized when the execution time among pipeline stages is well-balanced, but it is not straightforward. Several partition methods are possible, but each of them has its own pros and cons. First, the model can be partitioned by balancing the execution time of layers per stage. By balancing the execution time of layers per stage, the size of the bubble is minimized. However, balancing the execution time of layers may lead to memory imbalance among pipeline stages. Second, stages can be partitioned by balancing the memory consumption of GPUs. Balancing the memory footprint across pipeline stages can provide advantages in memory-constrained environments. However, because of the imbalance of execution time among stages, the size of the bubble increases. We evaluate how the stage partition methods affect the energy saving in § 6.2.3 (Table 2).

Bubble-reducing methods. Our work stands apart from previous studies that aim to reduce bubbles in S-PP [15, 16, 20]. While these studies may reduce pipeline bubbles, the bubbles cannot be completely eliminated, so EnvPipe can still leverage the bubbles to save energy. Moreover, these bubble-reducing methods have inherent drawbacks. For instance, Merak [15] necessitates activation recomputation, leading to additional performance overhead by repeated computations. Chimera [16] proposes a bidirectional pipeline but with additional memory consumption from weight parameters and activations since a single stage needs to maintain weights and activations that were originally maintained by two stages. PTD-T [20] introduces an interleaved 1F1B pipeline schedule, which reduces bubbles but adds communication overhead to the scheduling process. In contrast, our approach in EnvPipe effectively utilizes bubbles without introducing these drawbacks, offering a clear advantage in saving energy in pipeline parallelism.

4.5.2 Energy Consumption of Bubbles

Readers may raise the following question, "Would a naïve approach that just reduces the power consumption of bubbles save more energy than EnvPipe?" Even if one hypothetically assumes that the power usage of bubbles is reduced to 0W, EnvPipe's approach still demonstrates superior energy savings. Figure 8 illustrates a simplified example comparing the naïve approach and EnvPipe. In the naïve approach, the total energy consumption is 8.75J, where the forward unit consumes 8.75J of energy while the bubble consumes 0J. On the

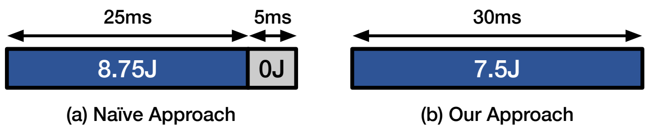


Figure 8: Energy consumption of bubbles

other hand, in ENVPIPE's approach where the forward unit is stretched to exploit the bubbles, the total energy consumption becomes 7.5J. The energy consumption of the stretched forward unit is already lower than the naïve approach's forward unit since decreasing the SM frequency reduces the end-to-end energy consumption of the forward unit. As demonstrated in this example, even when the power usage of bubbles is reduced to 0W, ENVPIPE's approach still has lower end-to-end energy consumption. Furthermore, in RTX3090, we measure that the power usage of bubbles is approximately 100W out of 350W at the lowest SM frequency with P2 P-State.

4.5.3 Scaling with Data Parallelism

The common practice for training large models that cannot fit in a single GPU is to combine pipeline parallelism with data parallelism or tensor parallelism, allowing for sharded models to fit within the constraints of a single GPU. We show in the evaluation section that our performance-preserving energy-saving approach with pipeline parallelism can easily scale with data parallelism in multi-node training.

5 Implementation

We implement our prototype, ENVPIPE, on top of DeepSpeed [25]. ENVPIPE's HW controller to lock SM frequency and DeepSpeed's basic mechanism such as executing forward/backward pass, send/receiving activations and gradients, and reducing computed gradients are used for executing ENVPIPE's performance-preserving energy-saving plan. In this section, we introduce some important considerations when utilizing the underlying mechanisms of ML platforms since naïve usage of those mechanisms can lead to ineffective usage of bubbles or performance degradation.

5.1 Asynchronous Communication

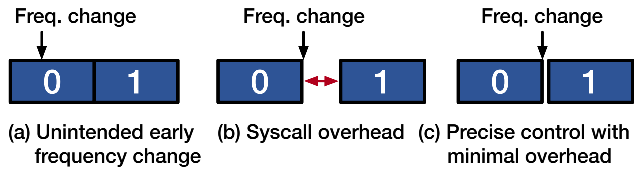
Activations and gradients must be transferred between GPUs for the pipeline to be executed. When sending activations or gradients, communication needs to be asynchronous (i.e. non-blocking) for pipeline units to effectively use bubbles. If GPU communication is synchronous, idle time of bubbles is wasted since blocking communication calls prevents the next scheduled pipeline units to execute. When the source GPU sends data to the target GPU, the communication gets blocked until the target GPU receives the data and the next scheduled pipeline units cannot be executed, thereby wasting the opportunity to utilize pipeline bubbles. On the other hand, for non-blocking asynchronous communication, when the source GPU sends data to the target GPU, the source GPU does not have to wait until the target GPU receives the data and can execute the next pipeline unit, effectively utilizing pipeline bubbles.

NCCL's default blocking p2p communication can be easily changed to non-blocking communication by increasing the NCCL buffer size with NCCL_BUFFSIZE environment variable. NCCL buffer is used when communicating data between pairs of GPUs. P2p send operation fills up the target GPU's buffer and the target GPU fetches data from the buffer in FIFO for another send operation to fill the buffer. If the NCCL buffer is full, send operation should wait until the buffer of target GPU has free space. If the NCCL buffer has enough free space, p2p send operation can complete without waiting for p2p recv operation to be called from target GPU. ENVPIPE makes sure that NCCL buffer size is enough to handle all activations and gradients to be communicated in a non-blocking way to effectively use bubbles.

5.2 Precise SM Frequency Control

Controlling SM frequency from user space needs an `ioctl` system call to the device driver which can induce overhead lowering the training throughput. Also because of the asynchronous CUDA programming model, `ioctl` calls on the CPU side should be executed with precise synchronization barriers between pipeline units for SM frequency to be controlled at exact timing.

Figure 9 shows how precise SM control with minimal overhead can be performed. In Figure 9(a), `lock_gpu_clock()` is called between two forward executions without any synchronization barrier. Because of the asynchronous CUDA programming model, `ioctl` call gets executed on the CPU side before the first forward execution completes, resulting in an unintended early frequency change. In Figure 9(b), the synchronization barrier is placed before `lock_gpu_clock()`,


Freq. change
0
1
(a) Unintended early frequency change
Freq. change
0
1
(b) Syscall overhead
Freq. change
0
1
(c) Precise control with minimal overhead

```
# (a) Unintended early frequency change
exec_forward_pass(0)
lock_gpu_clock(1300)
exec_forward_pass(1)
```

```
# (b) System call overhead
exec_forward_pass(0)
torch.cuda.synchronize()
lock_gpu_clock(1300)
exec_forward_pass(1)
```

```
# (c) Precise control with minimal overhead
exec_forward_pass(0)
torch.cuda.synchronize()
threading.Thread(lock_gpu_clock, args=(1300))
exec_forward_pass(1)
```

Figure 9: Precise SM frequency control

	Model	Microbatch	Minibatch
Single-V100	BERT-336M	4	64
	GPT-125M	2	32
	Megatron-125M	4	64
	ResNet-152	2	32
Single-3090	BERT-1.3B	4	64
	BERT-3.9B	2	32
	GPT-350M	4	64
	Megatron-350M	4	64
	Megatron-760M	4	64

Table 1: Model configuration for single-node training.

preventing unintended early frequency change. However, the `ioctl` call on the CPU side with the default synchronous programming model delays subsequent forward execution calls and GPU remains idle until the `ioctl` call returns. In Figure 9(c), by executing `lock_gpu_clock()` in another thread, subsequent forward execution on the GPU side can be concurrently executed with the `ioctl` call on the CPU side, performing precise frequency control with a minimal performance impact.

6 Evaluation

Our evaluations focus on confirming our energy-saving design choices that preserve DNN training performance. Specifically, we compare performance and energy consumption for various scheduling strategies when training different, up-to-date DNN models, including transformer-based language models and CNN models, on both single-node and multi-node training setups. To faithfully demonstrate the benefits of our approach in challenging scenarios, we have all models partitioned to balance execution time among pipeline stages, e.g., balancing the number of layers for transformer-based models. This pipeline-balanced parallel execution provides high training performance, making the energy savings on top of it more meaningful.

6.1 Methodology

Testbed setup. All experiments are performed on PyTorch 1.13. For single-node experiments, we use two different server setups: Single-V100 and Single-3090. **Single-V100** uses AWS `P3.8xLarge` instance which has four NVIDIA Tesla V100 GPUs with 16GB of memory each, Intel Xeon E5-2686 v4 CPU (32 vCPUs), and 244GB of main memory, while **Single-3090** has four NVIDIA Ampere RTX3090 GPUs with 24GB of memory each, Intel(R) Xeon(R) Gold 6326 CPU (64 vCPUs), and 256GB of main memory. For multi-node experiments, our setup **Multi-V100** uses two AWS `P3.16xLarge` instances each with 8 NVIDIA Tesla V100 GPUs, Intel Xeon E5-2686 v4 CPU (64 vCPUs), and 488GB of main memory. So, Multi-V100 is equipped with a total of 16 V100 GPUs. These cloud instances are connected to a 25Gbps Ethernet network.

Benchmarks. We compare EnvPipe with the following energy-saving methods:

- Baseline: run all GPUs with maximum SM frequency.
- Uniform: run all GPUs with optimal SM frequency that represents the minimum point in the energy valley curve.
- NoRecfg: EnvPipe without reconfiguring critical path to minimize performance impact.

Our single-node experiments are based on nine different models, as shown in Table 1. All models run with 16 microbatches, while for Megatron-125M, ResNet-152, and BERT-3.9B, the minibatch and microbatch size are reduced to fit in GPU memory. For multi-node experiments, we combine data parallelism (DP) and pipeline parallelism (PP) in several different ways across 16 GPUs. The number of microbatches for pipelining changes according to the DP and PP dimensions while the size of minibatch remains constant.

Metrics. Performance for all experiments is measured by averaging the throughput of 30 training iterations after warm-up and energy consumption is measured using the NVML library [4] during the 30 training iterations.

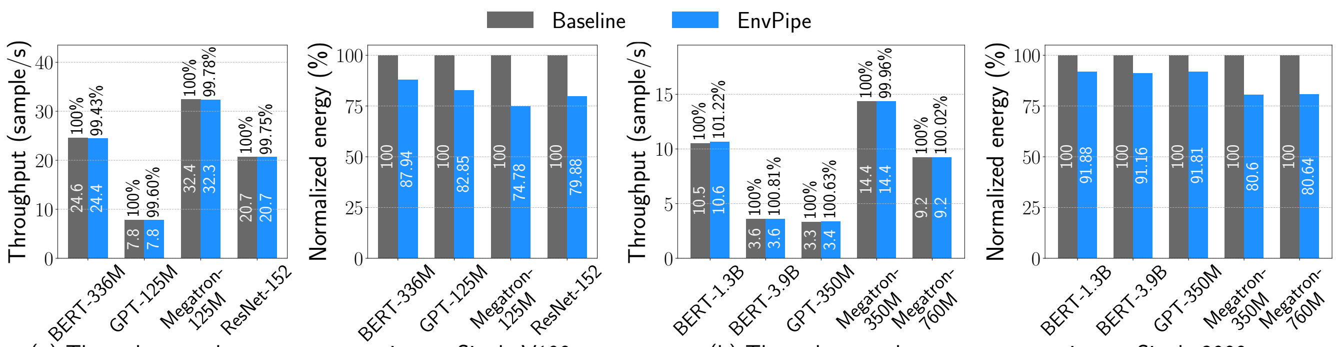
6.2 Experimental Results

6.2.1 Single-node Energy Saving

Main results. We first compare EnvPipe to Baseline, which consumes the most energy among competing methods, to highlight our ability to preserve performance. We measure the throughput and energy consumption for the benchmarks in Table 1 and present the results in Figure 10. The results show that EnvPipe consistently uses less energy than Baseline while retaining the original training performance for all models, confirming its effectiveness. Specifically, for Single-V100, EnvPipe saves energy consumption by an average of 18.6%, ranging from 12.1% to 25.2%. For Single-3090, EnvPipe saves energy consumption by an average of 12.8%, ranging from 8.1% to 19.4%. Performance is mostly preserved, with throughput only degrading less than 1% in all cases.

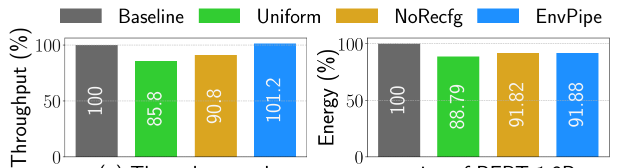
Analysis of energy saving. We next compare EnvPipe to other baselines by focusing on two models, BERT-1.3B and GPT-350M, trained on Single-3090. The results from Figure 11 show that other naïvely designed strategies, such as Uniform or NoRecfg, fail to meet our energy savings goal without sacrificing performance. Specifically, in Uniform, as all GPUs blindly scale SM frequency without considering performance effects, both throughput and energy consumption inevitably become the lowest. In NoRecfg, which does not reconfigure the SM frequency, all pipeline units inside the outer envelope are stretched to the largest extent. This strategy also inevitably degrades performance since not all bubbles can accommodate those stretched pipeline units inside the envelope. On the other hand, EnvPipe allows for more adaptive and effective use of pipeline bubbles, minimizing the impact on performance.

[4]Energy consumption of GPUs (not entire system).

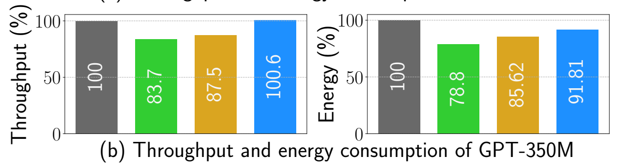


(a) Throughput and energy consumption on Single-V100 (b) Throughput and energy consumption on Single-3090

Figure 10: Throughput and energy consumption of various DNN models in single-node training



(a) Throughput and energy consumption of BERT-1.3B



(b) Throughput and energy consumption of GPT-350M

Figure 11: Normalized throughput and energy consumption breakdown

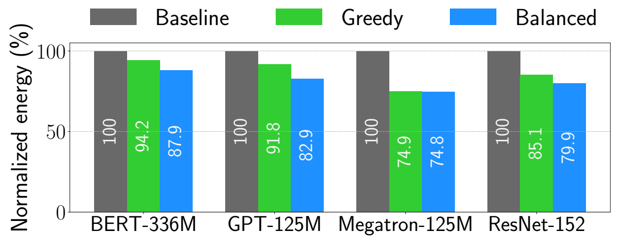
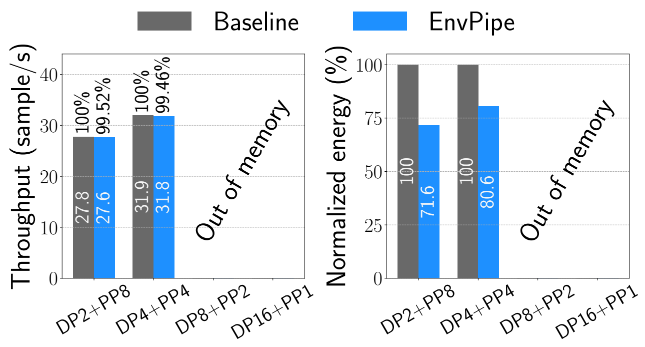


Figure 12: Comparison of reconfiguration policy

Comparison of reconfiguration policies. We now evaluate two different reconfiguration policies for the frequency planner of ENVPIPE, balanced (our default) and greedy, using models trained on Single-V100. The results shown in Figure 12 demonstrate that for BERT-336M, GPT-125M, and ResNet-152, the balanced method saves more energy than the greedy method, with a range of 5.2 to 8.9%. However, the difference in energy savings is insignificant for Megatron-125M. This is because the portion of pipeline bubbles (a measure of how much time is spent by bubbles) is larger for Megatron models due to additional computation on the loss computation layer, which is insignificant in other models, leading to longer execution times in the last stage of the pipeline. This portion exists even though the model is evenly partitioned across GPUs w.r.t. the number of transformer-based layers placed on each GPU. In summary, the balanced method delivers more benefits than the greedy method but exhibits different relative effectiveness according to how bubbles are composed.



DP	PP	Microbatch	Minibatch	Num of Microbatch
2	8	2	512	128
4	4	2	512	64
8	2	2	512	32
16	1	2	512	X

Figure 13: Throughput and energy consumption of Megatron-1.3B on Multi-V100

6.2.2 Multi-node Energy Saving

To study the efficacy of ENVPIPE under multi-node training, we examine the throughput and energy consumption of Megatron-1.3B trained on Multi-V100. Training sweeps different data parallel (DP) and pipeline parallel (PP) dimensions, as shown in Figure 13. Achieving efficient memory utilization in distributed training can be challenging when relying solely on data parallelism. Due to memory constraints, training could not be completed for DP8+PP2 and DP16+PP1. Also, we omit DP1+PP16, i.e., single-way DP combined with 16-way PP, since it is challenging to evenly split 24 transformer-based layers in Megatron-1.3B over 16 pipeline stages. We thus compare the throughput and energy consumption of Baseline and ENVPIPE mainly for DP2+PP8 and DP4+PP4, and show the results in Figure 13. ENVPIPE saves energy by 28.4% for DP2+PP8 and 19.4% for DP4+PP4 compared to Baseline. Similar to the single-node experiments, the throughput degradation for both cases is less than 1%, indicating that ENVPIPE can maintain its benefits when scaling to two or potentially more GPU nodes.

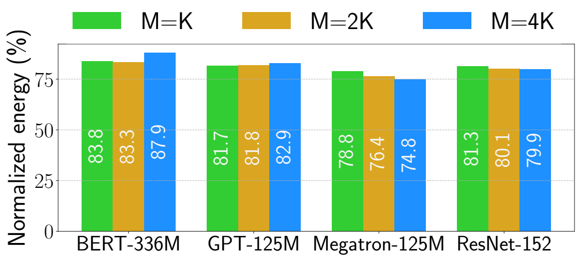


Figure 14: Sensitivity study of number of microbatches (M = number of microbatches, K = number of pipeline stages)

6.2.3 Sensitivity Study

Different number of microbatches. We examine how changing the number of microbatches affects the effectiveness of EnvPipe on Single-V100. The results, shown in Figure 14, reveal that the impact on energy savings differs between Megatron-125M and other models (BERT-336, GPT-125M, and ResNet-152). Megatron-125M model can save more energy when trained on a larger number of microbatches. This is largely due to an imbalance in pipeline stages and a larger portion of pipeline bubbles as compared to other models, which can be effectively harnessed over a long sequence of pipelined executions. However, for other models, increasing the number of microbatches does not result in further energy savings because it quickly leads to amortized bubble sizes.

Stage partition methods. Transformer-based models, which we have used for experiments thus far, are easy to partition in a balanced manner because of their regular structures. However, models like CNNs have non-regular structures, so different partitioning methods with EnvPipe can presumably offer different benefits. To investigate this, we evaluate two different partitioning methods, balancing execution time or memory footprint, using a popular CNN model, ResNet-152 on Single-3090. Both methods are considered beneficial from the perspective of system utilization, as they make training faster or more memory-efficient. The results from Table 2 show that balancing execution time leads to higher throughput, but the energy savings between the two methods are similar. This is because the stages at the front of the pipeline have more activations to store in memory, so shifting computation to the back of the pipeline stage can balance memory usage but cause an imbalance in computation between stages. Consequently, there is not much computation left to stretch in the pipeline stages at the front, limiting energy-saving opportunities.

7 Related Work

Pipeline parallelism. Bubbles in pipeline parallelism have been considered as an obstacle that slows down training throughput and previous studies focused on reducing the bubble with new scheduling methods [15, 16, 20]. On the other hand, EnvPipe considers bubbles as an opportunity and leverages bubbles to save energy.

Data center energy analysis. Recent studies focused on analyzing carbon emission and energy usage to measure the environmental impact when training large models in datacenters [14, 22, 23]. Treehouse [4] aims to reduce the carbon intensity of datacenters from a software perspective by providing suites of resources to application developers to better understand the trade-off between performance and carbon emissions. Strubell *et al.* [26] emphasizes the importance of quantifying the environmental cost of training neural network models for NLP.

Partition Method	GPU Memory (GB)	Perf. (sample/s)	Energy (%)
Execution Time	7.5 / 7.1 / 6.1 / 3.8	20.5	83.1
Memory	6.9 / 5.0 / 6.0 / 5.0	14.2	84.4

Table 2: Comparison of partition method of pipeline stages

Improving energy efficiency with GPU DVFS. Previous approaches to improve energy efficiency in GPUs have utilized GPU DVFS techniques by characterizing the relationship between performance and energy efficiency [6, 9, 13, 28]. Tang *et al.* [27] studied the energy scaling trend of various DNN training jobs on modern datacenter GPUs. Zeus [29] considers batch size as a new control knob for improving energy efficiency on DNN training, navigating the energy-performance tradeoff with Pareto optimality. Similarly, Batchsizer [17] considers batch size on DNN inference. Unlike previous approaches, EnvPipe saves energy by leveraging only the side-effect-free control knob in multi-GPU training.

8 Conclusion

In this study, we propose EnvPipe, a performance-preserving energy-saving DNN training framework. EnvPipe saves energy with no accuracy and performance degradation by leveraging bubbles in pipeline parallelism. EnvPipe profiles the optimal SM frequency of each pipeline stage, schedules pipeline units to make the best out of usable bubbles, and selectively reduces the SM frequency of pipeline units as much as possible with only a certain limit to avoid any adverse performance delay. EnvPipe can save energy up to 25.2% energy in single-node training with 4 GPUs and 28.4% in multi-node training with 16 GPUs with less than 1% of performance degradation.

Acknowledgments

We thank the anonymous reviewers and our shepherd for their valuable comments and feedback. This work was supported by Samsung Electronics, Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. 2021-0-00871, Development of DRAM-Processing-In-Memory Chip for DNN Computing, 50%), the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. RS-2023-00211901), and Electronics and Telecommunications Research Institute (ETRI) grant funded by the Korean government (23ZS1300).

References

[1] Deepspeed: Extreme-scale model training for everyone. https://www.microsoft.com/en-us/research/blog/deepspeed-extreme-scale-model-training-for-everyone/. Accessed: 2023-01-12.

[2] Measuring greenhouse gas emissions in data centres: the environmental impact of cloud computing. https://www.climatiq.io/blog/measure-greenhouse-gas-emissions-carbon-data-centres-cloud-computing. Accessed: 2023-01-12.

[3] Yuki Abe, Hiroshi Sasaki, Shinpei Kato, Koji Inoue, Masato Edahiro, and Martin Peres. Power and performance characterization and modeling of gpu-accelerated systems. In *2014 IEEE 28th International Parallel and Distributed Processing Symposium*, pages 113–122, 2014.

[4] Thomas E. Anderson, Adam Belay, Mosharaf Chowdhury, Asaf Cidon, and Irene Zhang. Treehouse: A case for carbon-aware datacenter software. *CoRR*, abs/2201.02120, 2022.

[5] Sanjith Athlur, Nitika Saran, Muthian Sivathanu, Ramachandran Ramjee, and Nipun Kwatra. Varuna: Scalable, low-cost training of massive deep learning models. In *EuroSys 2022*, April 2022.

[6] Srikant Bharadwaj, Shomit Das, Yasuko Eckert, Mark Oskin, and Tushar Krishna. Dub: Dynamic underclocking and bypassing in nocs for heterogeneous gpu workloads. In *Proceedings of the 15th IEEE/ACM International Symposium on Networks-on-Chip*, NOCS '21, page 49–54, New York, NY, USA, 2021. Association for Computing Machinery.

[7] Robert A. Bridges, Neena Imam, and Tiffany M. Mintz. Understanding gpu power: A survey of profiling, modeling, and simulation methods. *ACM Comput. Surv.*, 49(3), sep 2016.

[8] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam M. Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Benton C. Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier García, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Díaz, Orhan Firat, Michele Catasta, Jason Wei, Kathleen S. Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. Palm: Scaling language modeling with pathways. *ArXiv*, abs/2204.02311, 2022.

[9] Rong Ge, Ryan Vogt, Jahangir Majumder, Arif Alam, Martin Burtscher, and Ziliang Zong. Effects of dynamic voltage and frequency scaling on a k20 gpu. In *2013 42nd International Conference on Parallel Processing*, pages 826–833, Oct 2013.

[10] Miro Hodak, Masha Gorkovenko, and Ajay Dholakia. Towards power efficiency in deep learning on data center hardware. In *2019 IEEE International Conference on Big Data (Big Data)*, pages 1814–1820, 2019.

[11] Yanping Huang, Yonglong Cheng, Dehao Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V. Le, and Zhifeng Chen. Gpipe: Efficient training of giant neural networks using pipeline parallelism. *CoRR*, abs/1811.06965, 2018.

[12] Y. Jiao, H. Lin, P. Balaji, and W. Feng. Power and performance characterization of computational kernels on the gpu. In *2010 IEEE/ACM Int'l Conference on Green Computing and Communications & Int'l Conference on Cyber, Physical and Social Computing*, pages 221–228, 2010.

[13] Toshiya Komoda, Shingo Hayashi, Takashi Nakada, Shinobu Miwa, and Hiroshi Nakamura. Power capping of cpu-gpu heterogeneous systems through coordinating dvfs and task mapping. In *2013 IEEE 31st International Conference on Computer Design (ICCD)*, pages 349–356, 2013.

[14] Alexandre Lacoste, Alexandra Luccioni, Victor Schmidt, and Thomas Dandres. Quantifying the carbon emissions of machine learning. *CoRR*, abs/1910.09700, 2019.

[15] Zhiquan Lai, Shengwei Li, Xudong Tang, Keshi Ge, Weijie Liu, Yabo Duan, Linbo Qiao, and Dongsheng Li. Merak: An efficient distributed DNN training framework with automated 3d parallelism for giant foundation models. *IEEE Transactions on Parallel and Distributed Systems*, 34(5):1466–1478, may 2023.

[16] Shigang Li and Torsten Hoefler. Chimera: Efficiently training large-scale neural networks with bidirectional pipelines. In *Proceedings of the International Conference for High Performance Computing, Networking,*

Storage and Analysis, SC '21, New York, NY, USA, 2021. Association for Computing Machinery.

[17] Seyed Morteza Nabavinejad, Sherief Reda, and Masoumeh Ebrahimi. Batchsizer: Power-performance trade-off for dnn inference. In *Proceedings of the 26th Asia and South Pacific Design Automation Conference*, ASPDAC '21, page 819–824, New York, NY, USA, 2021. Association for Computing Machinery.

[18] Deepak Narayanan, Aaron Harlap, Amar Phanishayee, Vivek Seshadri, Nikhil R. Devanur, Gregory R. Ganger, Phillip B. Gibbons, and Matei Zaharia. Pipedream: Generalized pipeline parallelism for dnn training. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, SOSP '19, page 1–15, New York, NY, USA, 2019. Association for Computing Machinery.

[19] Deepak Narayanan, Amar Phanishayee, Kaiyu Shi, Xie Chen, and Matei Zaharia. Memory-efficient pipeline-parallel dnn training. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 7937–7947. PMLR, 18–24 Jul 2021.

[20] Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, Amar Phanishayee, and Matei Zaharia. 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*, SC '21, New York, NY, USA, 2021. Association for Computing Machinery.

[21] NVIDIA. NVIDIA Management Library (NVML). `https://developer.nvidia.com/nvidia-management-library-nvml`.

[22] David Patterson, Joseph Gonzalez, Urs Hölzle, Quoc Le, Chen Liang, Lluis-Miquel Munguia, Daniel Rothchild, David R. So, Maud Texier, and Jeff Dean. The carbon footprint of machine learning training will plateau, then shrink. *Computer*, 55(7):18–28, 2022.

[23] David A. Patterson, Joseph Gonzalez, Quoc V. Le, Chen Liang, Lluis-Miquel Munguia, Daniel Rothchild, David R. So, Maud Texier, and Jeff Dean. Carbon emissions and large neural network training. *CoRR*, abs/2104.10350, 2021.

[24] Aurick Qiao, Sang Keun Choe, Suhas Jayaram Subramanya, Willie Neiswanger, Qirong Ho, Hao Zhang, Gregory R Ganger, and Eric P Xing. Pollux: Co-adaptive cluster scheduling for goodput-optimized deep learning. In *15th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 21)*, 2021.

[25] Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '20, page 3505–3506, New York, NY, USA, 2020. Association for Computing Machinery.

[26] Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for deep learning in NLP. *CoRR*, abs/1906.02243, 2019.

[27] Zhenheng Tang, Yuxin Wang, Qiang Wang, and Xiaowen Chu. The impact of gpu dvfs on the energy and performance of deep learning: An empirical study. In *Proceedings of the Tenth ACM International Conference on Future Energy Systems*, e-Energy '19, page 315–325, New York, NY, USA, 2019. Association for Computing Machinery.

[28] Farui Wang, Weizhe Zhang, Shichao Lai, Meng Hao, and Zheng Wang. Dynamic GPU energy optimization for machine learning training workloads. *IEEE Transactions on Parallel and Distributed Systems*, pages 1–1, 2022.

[29] Jie You, Jae-Won Chung, and Mosharaf Chowdhury. Zeus: Understanding and optimizing GPU energy consumption of DNN training. In *USENIX NSDI*, 2023.

[30] Lianmin Zheng, Zhuohan Li, Hao Zhang, Yonghao Zhuang, Zhifeng Chen, Yanping Huang, Yida Wang, Yuanzhong Xu, Danyang Zhuo, Eric P. Xing, Joseph E. Gonzalez, and Ion Stoica. Alpa: Automating inter- and Intra-Operator parallelism for distributed deep learning. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pages 559–578, Carlsbad, CA, July 2022. USENIX Association.