A Quantitative Analysis of Vocabulary Layers

Following the calculations of Narayanan et al. (2021) and neglecting insignificant terms, we present the computational and memory expenses in relation to a single transformer layer in Table 4. In this table we denote the microbatch size as b, sequence length as s, hidden dimension as h and vocabulary size as V. It is worth noting that the activation memory is excluded from this analysis, as it typically has a transient nature for vocabulary layers.

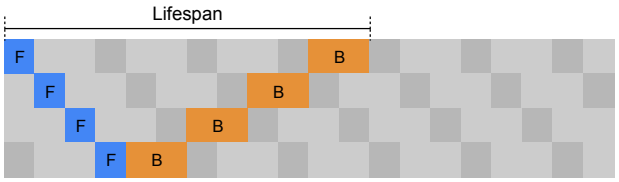
Table 4. Compute and memory cost of vocabulary and transformer layers

Layer Type	Compute FLOPs	Param Memory
Transformer	$bsh(72h + 12s)$	$24h^2$
Input	$3bsh$	$2hV$
Output	$6bshV$	$2hV$

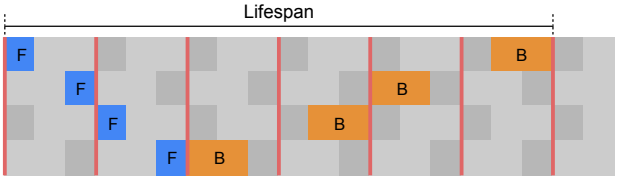
B More Analysis of Interlaced Pipeline

B.1 Memory Analysis

One concern of interlaced pipeline is that the peak activation memory of 1F1B is raised to 1.5 times of its original value. This increase in memory consumption can be analyzed using the framework introduced in Qi et al. (2024). As shown in Figure 15, the interlaced schedule enlarges the lifespan of 1F1B's building block from $3p$ to approximately $4.5p$ where p is the number of devices, resulting in 1.5x peak activation memory consumption.



(a) Building block of 1F1B



(b) Building block of interlaced pipeline parallel. The red vertical lines indicate synchronization introduced by TP of vocabulary layers.

Figure 15. Comparison between building blocks of 1F1B and Interlaced PP.

B.2 Overhead of Tensor Parallel Communication

In practice, tensor parallel is only used for intra-node parallelism due to its high communication volume. The interlaced pipeline has introduced a tensor parallel style parallelization for the vocabulary layers, which creates additional pipeline bubbles for each microbatch during the tensor parallel communication. To quantify the size of the bubbles, we conduct an ablation study by training a 21.5B model using 32 GPUs. We remove the synchronous all-reduce communications in the vocabulary layers, and measure the speedup in end-to-end iteration time. Note that the all-reduce communications that are overlapped with the computation are still kept.

By removing the synchronous all-reduce communications, the end-to-end iteration time improved by 10.95%. This shows that the synchronous all-reduce communications contributed to approximately 11% of the idle time when training using an interlaced pipeline. We conclude that the interlaced pipeline is undesirable for multi-node training.

C Vocabulary Parallelism for The Input Layer

While the output layers involve complex dependencies and communications, input layer computation can be completed independently before and after the transformer layer passes. The only required communications are an all-reduce communication after the forward pass, and a broadcast communication before the backward pass. These communications can be overlapped with the transformer layer computation, and can be scheduled well-ahead or after.

We schedule the input layer passes as follows:

- During the warm-up forward passes, we insert the input layer forward pass one microbatch before the first transformer layer forward pass. This allows time for gathering the input layer outputs.
- In the stable phase, we piggyback the input layer forward pass with the output layer passes, scheduled as least one repeating interval beforehand. Similarly, the input layer backward passes are piggybacked at least one repeating interval afterwards, allowing enough time to broadcast the output gradient to all devices.
- During the cool-down backward passes, we insert the input layer backward pass one microbatch after the last transformer layer backward pass.

This schedule ensures that each device is holding the input layer outputs for at most two microbatches at any instant, reducing the memory pressure.

D *V-Half* Pipeline Scheduling

Following the scheduling methodology in section 5.2, we show the building block for the *V-Half* schedule in Figure 16.

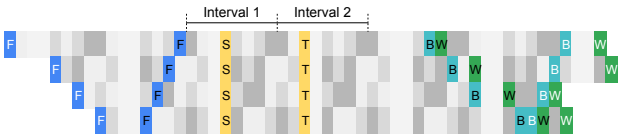


Figure 16. Modified building block for the *V-Half* schedule.

E Correctness Evaluation

Our implementation is based on the open-source Megatron-LM project (Narayanan et al., 2021). We compare the convergence curves of our implementation to that of the original Megatron-LM codebase to verify our implementation's correctness. The configurations follow the 4B model in section 6.2 with a vocabulary size of 256K, trained with 8 GPUs. Additionally, we verify that our implementation also works correctly with tensor parallelism, by using a pipeline parallel size of 4 and a tensor parallel size of 2.

The convergence curves are shown in Figure 17. It is shown that our implementation maintains correctness, albeit with some small numerical differences.

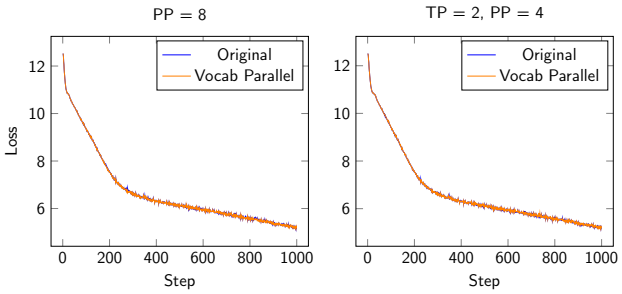


Figure 17. Convergence curves of our implementation against the original Megatron-LM codebase

F Detailed Experiment Data

For Sections 6.3 and 6.4, we present the detailed experimental data in Tables 5 and 6 respectively. The following metrics are computed:

- MFU: The FLOPs utilization of the training system. We follow Narayanan et al. (2021)'s derivation to compute the FLOPs of the model.
- Peak Memory: The maximum peak memory across all devices.

Table 5. Comparison of Methods on 1F1B.

Setup	Method	MFU (%)				Peak Memory (GB)			
		32k	64k	128k	256k	32k	64k	128k	256k
8GPU, Seq Length 2048	Baseline	46.16	40.48	33.11	25.23	14.86	16.32	19.25	25.64
	Redis	46.01	46.37	44.22	38.91	14.86	16.32	19.25	25.64
	Vocab-1	50.42	50.28	49.93	50.12	15.63	16.02	16.84	18.59
	Vocab-2	50.23	50.18	49.82	49.69	**14.83**	**15.23**	**16.04**	**17.78**
	Interlaced	**51.18**	**50.94**	**50.97**	**50.92**	17.20	17.57	18.43	20.17
8GPU, Seq Length 4096	Baseline	47.05	41.87	35.00	26.75	**21.39**	**22.85**	25.78	31.64
	Redis	46.93	46.78	47.44	43.01	**21.39**	**22.85**	25.78	31.64
	Vocab-1	50.98	50.98	50.83	50.66	24.04	24.47	25.41	27.34
	Vocab-2	50.93	50.75	50.56	50.40	22.44	22.89	**23.80**	**25.73**
	Interlaced	**51.41**	**51.82**	**51.32**	**51.38**	27.20	27.64	28.60	30.53
16GPU, Seq Length 2048	Baseline	45.66	40.09	32.44	24.21	24.03	25.98	29.92	38.71
	Redis	45.56	42.82	38.65	36.98	24.03	25.98	29.92	38.71
	Vocab-1	**49.02**	**50.62**	**50.54**	**50.66**	24.37	24.63	25.14	26.26
	Vocab-2	48.90	50.49	50.46	50.46	**23.57**	**23.83**	**24.35**	**25.47**
	Interlaced	48.94	48.97	49.19	49.52	29.23	29.47	29.97	31.10
16GPU, Seq Length 4096	Baseline	47.56	41.21	33.88	25.33	**36.99**	38.94	42.85	50.90
	Redis	47.41	43.07	43.15	40.15	**36.99**	38.94	42.85	50.90
	Vocab-1	50.93	**50.97**	**50.71**	**51.22**	39.46	39.73	40.31	41.53
	Vocab-2	**50.97**	50.80	50.68	50.90	37.89	**38.18**	**38.77**	**39.92**
	Interlaced	49.52	49.53	49.77	49.84	49.16	49.44	50.05	51.28
32GPU, Seq Length 2048	Baseline	42.81	37.28	28.97	20.86	33.45	35.89	41.17	52.16
	Redis	43.48	37.29	36.32	29.16	33.45	35.89	41.17	52.16
	Vocab-1	**45.85**	**45.92**	**45.90**	46.11	33.38	33.55	33.86	34.51
	Vocab-2	45.54	45.86	45.86	**46.16**	**32.72**	**32.88**	**33.20**	**33.84**
	Interlaced	42.40	42.43	42.75	43.25	42.94	43.09	43.40	44.07
32GPU, Seq Length 4096	Baseline	43.68	38.11	30.05	21.63	**54.97**	57.41	62.29	73.05
	Redis	44.01	38.12	37.87	31.03	**54.97**	57.41	62.29	73.05
	Vocab-1	**46.41**	**46.44**	**46.68**	46.83	57.41	57.56	57.88	58.58
	Vocab-2	46.23	46.35	46.55	**46.84**	56.09	**56.26**	**56.61**	**57.31**
	Interlaced	-	-	-	-	-	-	-	-

Table 6. Comparison of Methods on *V-Half*.

Setup	Method	MFU (%)				Peak Memory (GB)			
		32k	64k	128k	256k	32k	64k	128k	256k
16GPU, Seq Length 2048	Baseline	46.41	38.52	28.75	19.99	15.57	19.77	28.55	46.77
	Vocab-1	**52.82**	**53.11**	**53.41**	**52.89**	**13.20**	**13.46**	**13.98**	**15.02**
16GPU, Seq Length 4096	Baseline	50.01	41.17	31.36	21.90	21.22	25.61	34.56	53.11
	Vocab-1	**58.69**	**58.56**	**58.44**	**57.59**	**20.14**	**20.41**	**20.96**	**22.06**
24GPU, Seq Length 2048	Baseline	51.07	43.13	32.38	22.54	23.94	29.12	39.98	61.71
	Vocab-1	**56.70**	**56.50**	**55.72**	**54.86**	**21.08**	**21.29**	**21.72**	**22.57**
24GPU, Seq Length 4096	Baseline	54.53	45.96	34.99	24.31	33.60	38.97	49.90	72.60
	Vocab-1	**60.09**	**60.09**	**59.42**	**58.22**	**32.55**	**32.78**	**33.22**	**34.12**
32GPU, Seq Length 2048	Baseline	52.80	45.56	35.69	-	34.11	40.28	53.22	-
	Vocab-1	**57.70**	**57.62**	**57.69**	**57.80**	**30.85**	**31.04**	**31.42**	**32.18**
32GPU, Seq Length 4096	Baseline	56.06	48.17	37.85	-	48.84	55.19	68.12	-
	Vocab-1	**60.10**	**60.14**	**60.72**	**59.82**	**47.99**	**48.19**	**48.59**	**49.38**

G Artifact Appendix

G.1 Abstract

This section will outline the setup and experimental workflow of **Balancing Pipeline Parallelism with Vocabulary Parallelism** conducted on a single server equipped with 8 A100 GPUs.

G.2 Artifact check-list (meta-information)

- **Algorithm:** Vocabulary Parallelism, Online Softmax
- **Program:** Not used
- **Compilation:** Nvidia nvcc version 12.4, already in the container
- **Transformations:** Not used
- **Binary:** will be compiled on a target platform
- **Data set:** Customized C4 hosted in Huggingface
- **Binary:** will be compiled on a target platform.
- **Run-time environment:** Ubuntu 20.04.6. Needs docker. Requires root.
- **Hardware:** Server with 8 Nvidia A100 GPUs 80GB HBM.
- **Run-time state:** Server is idle.
- **Execution:** Takes at least 4 hours to complete.
- **Metrics:** Peak Memory, MFU
- **Output:** The data printed on console. The output of Quick Experiment is the key result.
- **Experiments:** Elaborated in the Installation and Experiment workflow sections.
- **How much disk space required (approximately)?:** 30 GB
- **How much time is needed to prepare workflow (approximately)?:** 30 minutes
- **How much time is needed to complete experiments (approximately)?:** 4 hours
- **Publicly available?:** Yes
- **Code licenses (if publicly available)?:** Apache License
- **Data licenses (if publicly available)?:** odc-by
- **Workflow framework used?:** No
- **Archived (provide DOI)?:**

G.3 Description

This experiment consists of 2 parts:

- **Quick Experiment** to quickly verify the result on a specific case.
- **Full Experiment** to run all cases on an 8-GPU server.

The **Full Experiment** employs the settings in table 7 same as the paper. The **Quick Experiment** focuses on a specific case where the sequence length is 4096 and the vocabulary size is 256k. We use Megatron-LM on which all methods are implemented to run training benchmarks.

Table 7. Artifact Settings used in experiments on 1F1B schedule.

Pipelines (GPUs)	8
Model Size	≈ 4B
Layers	32
Attention Heads	24
Hidden Size	3072
Sequence Length	2048 / 4096
Microbatch Size	1
Number of Microbatches	128
Vocabulary Size	32k / 64k / 128k / 256k

G.3.1 How delivered

The code locates on https://github.com/sail-sg/VocabularyParallelism. The dataset will be automatically downloaded in experiment scripts.

G.3.2 Hardware dependencies

The tests should be conducted in a server with 8 Nvidia A100 GPUs, 80GB HBM.

G.3.3 Software dependencies

- **CUDA Driver Version**: 535.54.03
- **CUDA Version** 12.2
- **Docker**
- **NVIDIA Container Toolkit**

G.3.4 Data sets

The dataset is customized from C4 hosted in Huggingface https://huggingface.co/datasets/mtyeung/vocab_parallel_sample_dataset supporting various sequence length. It will be automatically downloaded in experiment scripts.

G.4 Installation

Run a container:

```
docker run --name vocab_torch24 \
    --network=host -d \
    --runtime=nvidia --gpus all \
    --ipc=host --ulimit memlock=-1 --ulimit
     stack=67108864 \
    --privileged=true \
    nvcr.io/nvidia/pytorch:24.03-py3 sleep
    infinity
```

Get inside the container, and clone the codes:

```
# Enter the container
docker exec -it vocab_torch24 bash
# Clone the codes
git clone https://github.com/sail-sg/
    VocabularyParallelism.git
cd VocabularyParallelism
```

Note that all the following commands should be run inside the `VocabularyParallelism` directory.

G.5 Experiment workflow

G.5.1 Quick Experiment

The quick experiment runs all the methods (*baseline*, *redis*, *interlaced*, *vocab-1*, *vocab-2*) on a specific setting in the paper:

- Sequence Length: 4096
- Vocabulary Size: 256k

The experiment will show 2 key results:

- **Peak Memory**
- **MFU**

Run all the methods one by one:

```
bash artifact/quick_exp.sh run baseline
bash artifact/quick_exp.sh run redis
bash artifact/quick_exp.sh run interlaced
bash artifact/quick_exp.sh run vocab-1
bash artifact/quick_exp.sh run vocab-2
```

This will automatically download the dataset from huggingface and run the training experiments. The log containing the result will locate in `quick-logs/<method>/stdout.log`. Each method should take around 6 minutes to complete.

Then run this to collect the results:

```
bash artifact/quick_exp.sh show-result
```

G.5.2 Full Experiment

This will run all experiments on a single server with 8 A100 GPUs.

The whole experiment will take around 3 hours to complete.

```
bash artifact/full_exp.sh artifact/
    exp_one_host.csv
```

Print results:

```
python artifact/show_result_full_exp.py
```

G.6 Evaluation and expected result

The output of **Quick Experiment** should show similar result as
https://github.com/sail-sg/VocabularyParallelism/blob/main/artifact/example-results/quick-exp.txt

The expected output of **Full Experiment** is located in
https://github.com/sail-sg/VocabularyParallelism/blob/main/artifact/example-results/full-exp.txt

The result should also roughly match the 2 rows in **Table 5. Comparison of Methods on 1F1B** in the paper:

- 8GPU, SEQ LENGTH 2048
- 8GPU, SEQ LENGTH 4096

The result shows that the throughput and peak memory of vocab-1 and vocab-2 are significantly better than baseline and redistribution. The throughput of interlaced is slightly better than vocab-1 and vocab-2. But the peak memory of interlaced is worse than our approach.

G.7 Experiment customization

User can customize the settings by changing the scripts quick_exp.sh, full_exp.sh, show_result_full_exp.py under under artifact/.