SA: Lab10 (k)

15. Optimization and Scaling of Deep LLM’s Training

Juan Jose Olivera y Pol Escola

December 2025

Universitat Politècnica de Catalunya (UPC)

Outline

1. Optimizing Throughput
    - 1-GPU Optimizations with Llama 3.2
    - Scaling to 32 GPUs with Llama 3.2
    - 1-GPU Optimizations with Facebook OPT
    - Scaling to 32 GPUs with Facebook OPT
2. Getting the Practical Benefits
    - Measuring Model Performance
    - Evaluating Performance of different optimized configurations

Task 15.0.1 1-GPU Optimizations

- **Batch Size:** Amount of samples processed simultaneously as a batch. Increased batch size usually improves throughput as implementations in GPU are optimized for this workflow. However, more batch size increases memory usage. Nvidia GPUs H100 have max 64GB per device.
- **Accelerated Mixed Precision (AMP):** Technique that transforms most numbers into lower precision (ex from FP32 to FP16 or BF16) to use less memory and increase computation (more numbers fit in an ALU).
- **Model Precision:** Technique used to change the model precision by setting the numeric representation of all of its parameters (lowering memory and conversions required to AMP).
- **Attention Mechanism:** Different implementations of attention mechanism. Crucial for Transformer models.
- **Liger Kernels:** Set of kernels fused and optimized for LLMs.
- **Torch Compilation:** Grouping of different GPUs kernels into a compute Graph for single launching.

Task 15.0.1 A Prior: Llama 3.2-1B 1-GPU Optimization

Naive FP32/NO AMP

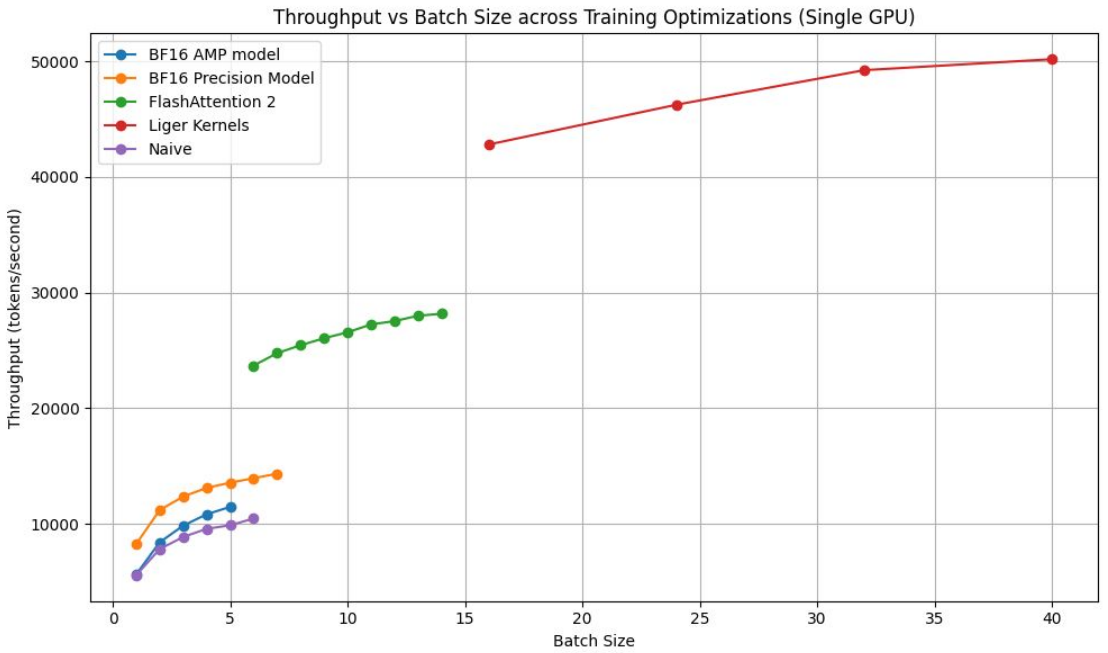
- Max Batch Size before OOM: 6
- Throughput: 10467.77 tokens/sec
- Memory: 61.9GB
- Epochs Covered: 0.27

Optimized BF16/BF16

- Max Batch Size before OOM: 37
- Throughput: 48833.13 tokens/sec
- Memory: 62.34GB
- Epochs Covered: 1.66

After 1-GPU Optimizations

Throughput Speedup: 4.67x



Task 15.0.2 A Prior: Llama 3.2-1B Scaling 32GPUs

Single GPU:

- GPU-Throughput: 48833.13 tokens/sec
- Global Throughput: 48833.13 tokens/sec
- Epochs: 0.27

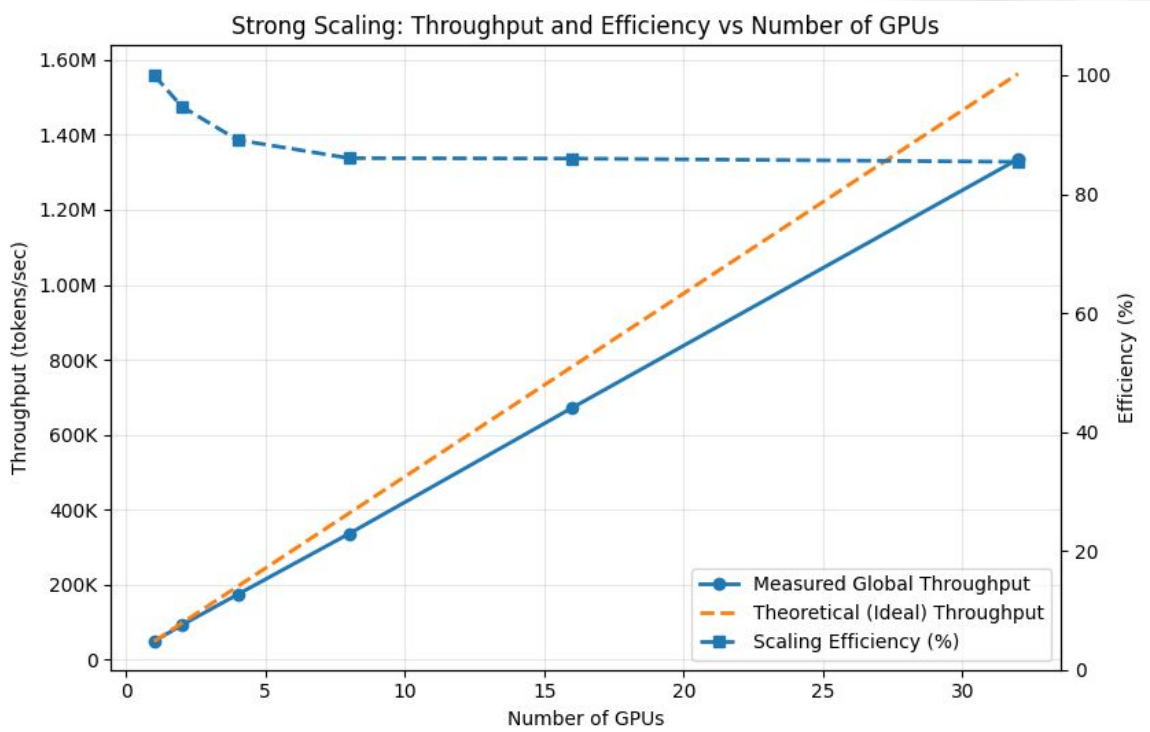
32-GPUs:

- GPU-Throughput: 41.7×10^3 tokens/sec
- Global Throughput: 1.3×10^6 tokens/sec
- Epochs: 56.25

Scaling to 32 GPUs

Throughput Speedup: 27.33x

Efficiency: 85.4%



Task 15.0.2 A Prior: Llama 3.2-1B Scaling 32GPUs

Llama 3.2 1B can be optimized and scaled SUPERB
Huge Optimizations:

- **BF16 AMP**
- **BF16 Precision Model**
- **Flash Attention**
- **Liger Kernels**

Reflection
Passing the model to BF16 precision/AMP had a small benefit (1.4x improvement).
However, FlashAttention 2 yield 2x improvement from BF16 precision/AMP.
And Liger Kernels another 1.79x.

What about other Models?

Task 15.1 – Reproducing Experiment with Facebook OPT 1B

Some characteristics:

Facebook OPT 1B	**Llama 3.2 1B**
~1.3 billion parameters	~1 billion parameters
Released in 2022	Released in 2024
Absolute Position Encoding	Rotary Position Encoding RoPE
Activation Function: ReLU	Activation Function: SwiGLU
Multihead Attention MHA	Grouped Query Attention GQA

Task 15.1.1 - Facebook OPT-1B 1-GPU Optimization

Naive FP32/NO AMP

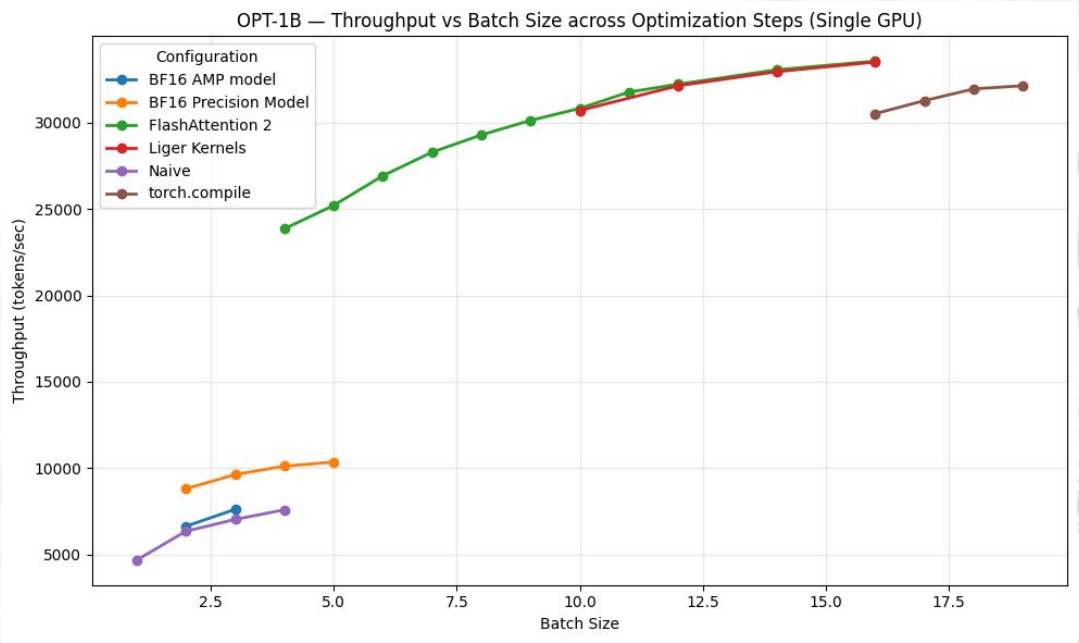
- Max Batch Size before OOM: 4
- Throughput: 7601.89 tokens/sec
- Memory: 57.60GB
- Epochs Covered: 0.18

Optimized BF16/BF16

- Max Batch Size before OOM: 16
- Throughput: 33544.73 tokens/sec
- Memory: 59.11GB
- Epochs Covered: 0.72

After 1-GPU Optimizations

Throughput Speedup: 4.41x



Task 15.1.2 - Facebook OPT-1B Scaling 32GPUs

Single GPU:

- GPU-Throughput: 33569.96 tokens/sec
- Global Throughput: 33569.96 tokens/sec
- Epochs: 0.72

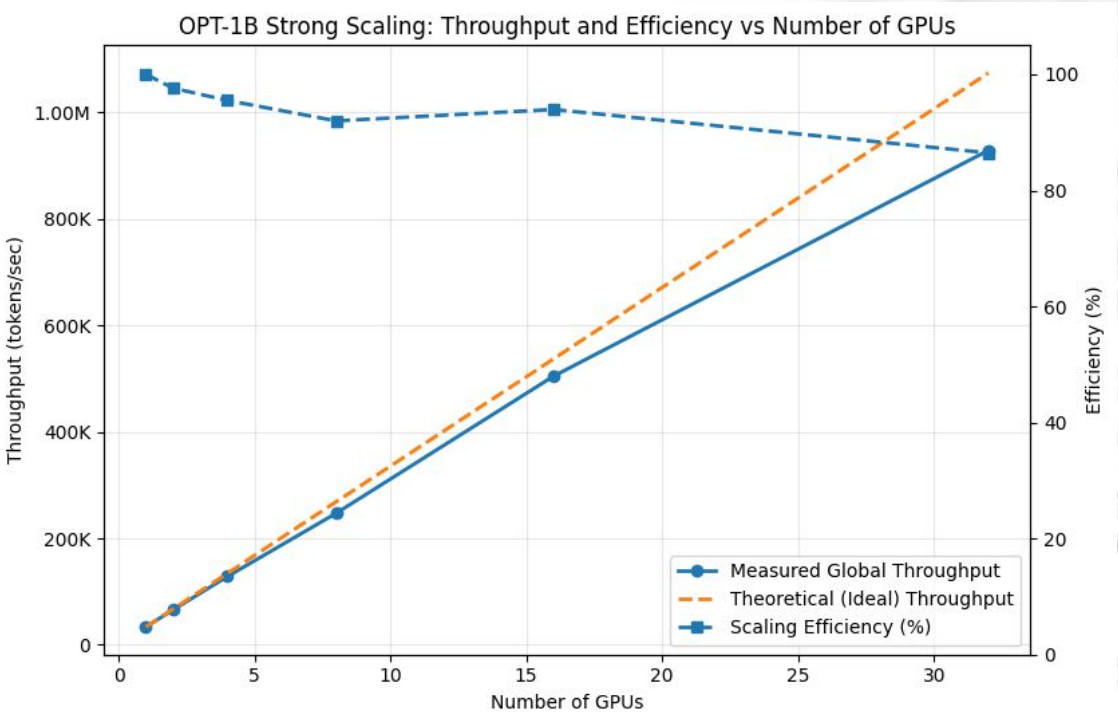
32-GPUs:

- GPU-Throughput: 2.9×10^3 tokens/sec
- Global Throughput: 928.83×10^3 tokens/sec
- Epochs: 22.5

Scaling to 32 GPUs

Throughput Speedup: 27.67x

Efficiency: 86.46%



Task 15.3-11

Llama 3.2 1B can be optimized and scaled SUPERB

Huge Optimizations:

- **BF16 AMP**
- **BF16 Precision Model**
- **Flash Attention (Most benefit)**
- **Liger Kernels (Didn't add benefit)**

Reflection:

Contrary to Llama 3.2-1B, Facebook OPT's Liger Kernels didn't have an effect (as seen in lines green and red) and BF16 Model Precision/AMP impact was smaller than with Llama.

Torch Compile did lower the memory footprint allowing more batches, but so it lowered the throughput, making it effectively similar when increasing up to OOM.

Facebook OPT-1B scalability with 32 GPUs was decent, obtaining slightly better efficiency, however, it is less adequate for the applied 1-GPU performance optimizations.

What about actually improving LLM accurate performance = Loss?

Problem With Loss

Conventional Wisdom: More Training => Better Model Accuracy = Lower Loss

However,
Final Loss remains equally bad (high loss) regardless of throughput increasing drastically.

Scenario	**Naive OPT Batch Size=5 in 1 GPU**	**Optimized OPT Batch Size=16 in 1 GPU**	**Optimized OPT Batch Size=16 in 32 GPUs**
Global Throughput	7.6K toks/sec	33.57K toks/sec	928.8K toks/sec
Training Time	242 seconds	219 seconds	254 seconds
Final Training Loss	**6.954**	**6.964**	**6.964**

Problem With Loss

Root Cause:
Training Dataset is a Dummy Random Dataset:

+ Good for Stressing Throughput Optimization
- Bad for Measuring Model Learning Effectiveness

```python
class DummyDataset(Dataset):
    def __init__(self, sequence_length: int = 8192, num_samples: int = 1000000) -> None:
        self.num_samples = num_samples
        self.sequence_length = sequence_length
    def __len__(self) -> int:
        return self.num_samples
    def __getitem__(self, idx: int) -> Dict[str, np.ndarray]:
        x = torch.LongTensor(np.random.randint(low= 0, high= 1000, size=(self.sequence_length + 1)))
        return {"input_ids": x[:-1], "labels": x[1:]}
```

Chapter.15/utils.py

Solution: Task 14 FineTuning Facebook OPT-1B on WikiText!

Getting the WikiText Dataset *huggingface-cli download Salesforce/wikitext*

```
itadmin@helgen:~/mojo$ huggingface-cli download "Salesforce/wikitext" --local-dir "./data" --repo-type dataset
Fetching 16 files:   0%|                                                                                | 0/16 [00:00<?, ?it/s]
Downloading 'wikitext-103-v1/train-00000-of-00002.parquet' to 'data/.cache/huggingface/download/wikitext-103-v1/U-Bqi1ccYPaG7kHCF5eNyOm2leU=.c2ecca8c3250e79
518e45d125f3a9a757d8014f6b2d8435c602be87c1f79ec3b.incomplete'
Downloading 'wikitext-103-raw-v1/train-00000-of-00002.parquet' to 'data/.cache/huggingface/download/wikitext-103-raw-v1/U-Bqi1ccYPaG7kHCF5eNyOm2leU=.74da360
f23826045b3e6ac6375411fdb15f003030aa74f2596ed08b857cb9212.incomplete'
Downloading 'wikitext-103-raw-v1/train-00001-of-00002.parquet' to 'data/.cache/huggingface/download/wikitext-103-raw-v1/4PAW2js2aor0N0glxWM5DEwzF3M=.ba090ac
50dbf5461e8dcbdd1a1b8e6f3cf9c2c756d64f0c1220450acd514f720.incomplete'
Downloading 'wikitext-103-v1/test-00000-of-00001.parquet' to 'data/.cache/huggingface/download/wikitext-103-v1/iln7FAK8ScOKT0OL8tHV8680tg8=.abdfc9f83b1103b5
02924072460d4c92f277c9b49c313cef3e48cfcf7428e125.incomplete'
Downloading 'wikitext-103-raw-v1/test-00000-of-00001.parquet' to 'data/.cache/huggingface/download/wikitext-103-raw-v1/iln7FAK8ScOKT0OL8tHV8680tg8=.5f1bea06
7869d04849c0f975a2b29c4ff47d867f484f5010ea5e861eab246d91.incomplete'
                                                                                                                D
Downloading 'README.md' to 'data/.cache/huggingface/download/Xn7B-BWUGOee2Y6hCZtEhtFu4BE=.2a4fec2bc8df76c9d4da1c8e8865b625eb221c76.incomplete'[00:00<?, ?B/s]
itadmin@helgen:~/mojo$ cd data/
itadmin@helgen:~/mojo/data$ ls
README.md  wikitext-103-raw-v1  wikitext-103-v1  wikitext-2-raw-v1  wikitext-2-v1
itadmin@helgen:~/mojo/data$
```

Task 14: FineTuning Model on WikiText!

Fine-tuning Facebook OPT-1B in MN5 with 10% of WikiText-2-raw-v1

```python
from transformers import AutoModelForCausalLM, AutoTokenizer

print('Loading model...')
model_name = "facebook/opt-1.3b"
model = AutoModelForCausalLM.from_pretrained("/gpfs/projects/nct_345/facebook-opt-1.3b")
tokenizer = AutoTokenizer.from_pretrained("/gpfs/projects/nct_345/facebook-opt-1.3b")


def generate_text(prompt, max_length=100):
  inputs = tokenizer(prompt, return_tensors="pt").to(model.device)
  output = model.generate(**inputs, max_length=max_length)
  return tokenizer.decode(output[0], skip_special_tokens=True)

# print('Generating text...')
# prompt = "What are the applications of quantum computing?"
# print(generate_text(prompt))

print('Try finetuning...')

from transformers import TrainingArguments, Trainer
from datasets import load_dataset

print('Loading training args')
training_args = TrainingArguments(
    output_dir="./fine-results",
    per_device_train_batch_size=4,
    num_train_epochs=1,
    report_to=None
)

print('Load training dataset')
dataset = load_dataset(
  path="./data",
  name="wikitext-2-raw-v1",
  split="train[:10%]"
)

tokenizer.pad_token = tokenizer.eos_token

def tokenize_function(examples):
```

We load model and tokenizer using **transformers** library functions from the facebook opt path

We load 10% of dataset from absolute path using **datasets** library

custom_llm_training.py

Task 14: FineTuning Model on WikiText!

Fine-tuning Facebook OPT-1B in MN5 with 10% of WikiText-2-raw-v1

```
tokenizer.pad_token = tokenizer.eos_token

def tokenize_function(examples):
  encoding = tokenizer(
    examples["text"],
    padding="max_length",
    truncation=True,
    max_length=512,
  )
  encoding["labels"] = encoding["input_ids"].copy()
  return encoding

print('Tokenizing dataset...')
tokenized_dataset = dataset.map(tokenize_function, batched=True)

print('Creating trainer')
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_dataset
)
print('Training...')
trainer.train()

print('Done.')
```

We tokenize for facebook opt 1B the dataset

We train with transformers trainer

custom_llm_training.py

Task 14: FineTuning Model on WikiText!

We rerun previous Facebook OPT-B runs but with WikiText dataset instead of DummyDataset.

Results: **FINAL LOSS REDUCES** AS WE OPTIMIZE AND INCREASE COMPUTE!!!!
0.4174 -> 0.1242 (close to 4 times better according to our knowledge/opinion)
(Throughput changes as WikiText dataset sample lengths varies in contrast to DummyDataset)

Scenario	Naive OPT Batch Size=5 in 1 GPU	Optimized OPT Batch Size=16 in 1 GPU	Optimized OPT Batch Size=16 in 32 GPUs
Global Throughput	15.6K toks/sec	58.39K toks/sec	1.5M toks/sec
Training Time	117 seconds	126 seconds	156 seconds
Final Training Loss	**0.4174**	**0.3455**	**0.1242**

End Remarks

For LLM training/finetunning, **FIRST optimize for 1 GPU**.
Increased over **4x to 5x times**. Even 10x if we considered worst naive BatchSize=1.

Scaling LLM training is very efficient and straightforward with Transformers package: **85% scaling efficiency = 27x.**

Newer models have better baseline performance and optimization strategies.

Throughput can be tested with DummyDatasets BUT model accuracy must be evaluated with REAL DATASETS.

Substituting with a real dataset (10% of WikiText-2-v1) we lower Loss from 0.42 to 0.14 in about same Wall-Clock Time. This is a prime example of WEAK SCALING.

THANK YOU

Compromesos
amb la qualitat dels
nostres serveis.

Ref: Torres, J. (2025). *Supercomputing for Artificial Intelligence*.