COMP3231 Assignment 2

Topic: Building and Analyzing a Pipelined Execution Engine for AI Training

Task 1: Pipeline Training on CPUs

Environment requirements:

- PyTorch Version >= 1.7.0
- Python Version >= 3.6.8

In each ***-template.py** file (included in the file `template.zip`), we provide a four-stage AI training template code.

1. Given a four-layer Convolutional Neural Network (CNN) model, the code partitions it into four stages. 2.
2. The code spawns five processes, with each assigned to one partition of the CNN model and one CPU core, except for the first one, which is designated as the master process.
3. The code executes each partition of the CNN model sequentially: each part of the model is trained on a separate CPU core.

However, as a CNN model training is executed sequentially by a forward pass and a backward pass, the template code suffers from heavy CPU under-utilization: at any time, only one CPU core is working. In this task, you are required to modify the template code to inject multiple executions and making the training of CNN pipelined.

To complete this task, we assume that you have a computer with at least **4 CPU cores**, running Ubuntu Linux or Windows Subsystem for Linux 2 (WSL2). The template code may not run on native Windows environments or macOS with Apple silicon. You can use the GPU farm (https://www.cs.hku.hk/gpu-farm/quickstart) to run the CPU-based template code.

Task 1.1 (20%): Implement the four-stage CPU version pipeline training engine.

- In this task, you need to make the execution pipelined, so that all four CPU cores can be fully activated at the same time. You can add more logging in your code to ensure that your pipeline runs correctly.
- Do not modify the CPU affinity.

Task 1.2 (30%): Partition two given popular DNNs (ResNet50, VGG16) into four stages for execution that achieve the shortest training time, according to pipeline hazards (from lecture 3, starting at slide 36).

- You should aim to create partitions that achieve the highest training throughput (i.e., the shortest execution time for each training batch).
- You can only consider the models as a sequence of connected layers, with each layer receiving input from only the previous layer.

- Present the evaluation results.
  - For each DNN model, display a table with the partition settings (the layer numbers for each stage) you have tested (at least three partition settings) and identify the partition that achieves the highest training throughput. For each partition, record the execution time of a single batch. You may observe that the first batch has (or the first a few batches have) a relatively slower processing speed. You can safely ignore this part when performing average calculations.
  - Increase the batch size from 1 to 4, 8, and 16, and document the variance in training execution time for ResNet50 with the chosen partition from the table above.

Task 1.3 (20%): Analysis and Report.

- What are the key factors that affect the training throughput?
- Generally speaking, what's the best (ideal) partition that can maximize the throughput of a training pipeline?
- What's the distance between your current partition on the two models and the ideal partition?

Task 2: Pipeline Training on GPUs

In this task, you are required to run pipeline training on two GPUs with vPipe (https://github.com/hku-systems/vpipe.git) in the GPU farm. Follow steps below to prepare the environment (we have also provided videos):

1. Install Anaconda following instructions in https://www.cs.hku.hk/gpu-farm/quickstart
2. Install required packages:

```
gpu-interactive
conda create -n pipeline python=3.8
conda activate pipeline
pip install torch==1.7.1 torchvision==0.8.2
pip install packaging boto3 h5py requests tqdm
git clone https://github.com/NVIDIA/apex.git
cd apex
git checkout 22.04-dev
export PATH=/usr/local/cuda-10.2/bin:$PATH
pip install -v --disable-pip-version-check --no-cache-dir --global-option="--cpp_ext"
 --global-option="--cuda_ext" ./
exit
```

3. Train a Transformer model on two GPUs:

```
srun --gres=gpu:2 --cpus-per-task=8 --pty bash
conda activate pipeline
git clone https://github.com/hku-systems/vpipe.git
cd vpipe/runtime/bert
cd data/hdf5_lower_case_1_seq_len_128_max_pred_20_masked_lm_prob_0.15_random_seed_123
45_dupe_factor_5
mkdir bookcorpus
```

```
cat xa* > bookcorpus/books_wiki_en_corpus_training.hdf5
cd ../..
pkill python  # only needed when you re-run the command below
python -m launch --nnodes 1 --node_rank 0 --nproc_per_node 2 main_with_runtime.py --d
ata_dir data/hdf5_lower_case_1_seq_len_128_max_pred_20_masked_lm_prob_0.15_random_see
d_12345_dupe_factor_5/bookcorpus --master_addr localhost --module vgpus\=2 --checkpoi
nt_dir output --partition vgpus\=2/gpipe.json --sync_mode asp --distributed_backend g
loo -b 8 --lr 0.000600 --lr_policy polynomial --weight-decay 0.000000 --epochs 20 --p
rint-freq 10 --verbose 0 --num_ranks_in_server 2 --config_path vgpus\=2/mp_conf.json
```

Below is the output format of vPipe:

```
Stage: [0] Epoch: [0][i/s]   Memory: m (mm)
Stage: [1] Epoch: [0][i/s]   Time: t (at)   Epoch time [hr]: … Memory: m (mm) Loss: …
```

where i=number of batches executed, s=total number of batches, m=current GPU memory usage, mm=maximum GPU memory usage, t=execution time (in seconds) of current batch and at=average execution time of all batches executed.

The file gpipe.json in directory runtime/bert/ vgpus=2 looks like:

```
{
  "partition": [26, 23],
  "recompute_ratio": [1, 1]
}
```

The first array defines the number of layers in each stage.

Task 2.1 (30%): Try three partition settings by changing the array in file gpipe.json. Make sure the sum of array elements equals 49. Show the average execution time of the first 100 batches for each partition setting.

Deliverables: For Task 1.1, submit two Python code files: resnet.py and vgg.py. For other tasks, submit a single PDF file. Submit a ZIP file that contains all above files.