CSYE7105 HW4

Instructor: Dr. Handan Liu

2020-12-06

PART 1

1. **Please list the general idea of a CUDA workflow. (hint: 6 steps) [6 pts]**

**Sol:** CUDA is a platform and programming model for CUDA-enabled GPUs. The platform exposes GPUs for general purpose computing. In CUDA programming, both CPUs and GPUs are used for computing. Typically, we refer to CPU and GPU system as host and device, respectively. CPUs and GPUs are separated platforms with their own memory space. Typically, we run serial workload on CPU and offload parallel computation to GPUs.

* 1. Allocate memory on both Host and Device.
  2. Initialize variables on Host.
  3. Copy data from Host to Device.
  4. Perform parallel computation on Device.
  5. Copy data back from Device to Host.
  6. Free memory on both Host and Device.

1. **Please indicate which types of memory are on-chip and off-chip in GPU memory? [6 pts]**

**Sol: *on chip*** memory means having storage on the processor chip to decrease latency for critical/ frequently used data. It has less latency (usually single cycle access time) and works as a cache for the main memory(DRAM).

Whereas, ***off chip*** memory has much more latency (5–15 cycles). DRAM is the preferred off chip memory technology which uses an altogether different process technology. It is much cheaper than SRAM.

* Local, Global, Constant, and Texture memory all reside off chip.
* Local, Constant, and Texture are all cached.

1. **Why deep neural network training is slow? List three or more main reasons. [6pts]**

**Sol:** The reasons for slow training in DNN are:

* Models, for big tasks are often trained with millions to billions of training examples.
* Large datasets often use DNNs with a larger number of parameters.
* Most popular technology to train DNNs is the first-order stochastic gradient descent(SGD) optimization technique, which is serial algorithm executed on multi-core CPU.
* Training a neural network involves using an optimization algorithm to find a set of weights to best map inputs to outputs where model weights are updated each iteration using the backpropagation of error algorithm.
* Amount of non-linearity between the layers.
* Amount of Gradient noise.

1. **List parallel methods for DNN training on a single machine and multiple machines. [5pts]**

**Sol:** Local parallel training: Model and Data are stored on a single machine.

Distributed parallel training: Store the data or model across multiple machines or

multiple CPUs

1. *Local parallel training*

* Multi- core processing : share the memory.

Use the cores to process multiple images at once each layer. This is an embarrassingly parallel process.

Use multiple cores to perform SGD of multiple mini batches in parallel.

* Use GPU for computationally intensive subroutines like matrix multiplication.
* Use both multi-core processing and GPU where all cores share the GPU and computationally.

1. *Distributed parallel training*

* Data parallelism: Data is distributed across multiple machines.to a single machine, it can be split across multiple machines.
* Model parallelism: If the model is too big to be fit into a single machine and not so much to fasten the training process.

1. **What type of communication of MPI is used for data parallel deep learning? And give 2 MPI\_ functions for instance. [6pts]**

**Sol:** Collective communication is used for data parallel deep learning.

*Types of Collective communication:*

* Synchronization: Blocks until all processes have reached a synchronization point.
* Data Movement(or Global communication): Broadcast, Scatters, Gather, All to All transmission of data across the communicator.
* Collective Communicator: One process from the communicator collects data from each process and performs an operation(min, max, add, multiply etc) on that data to compute a result.

*2 MPI\_functions for instance*:

* MPI\_Bcast(GPU 0) -> Short for “broadcast”. Takes data from one node and sends it to all processes in the processes group.
* MPI\_Reduce(GPU 0) -> Takes data from all processes in a group, performs an operation(such as summing), and stores the results on one node.

1. **In deep learning, how many dimensional tensors are used to express the time series,**

**Images and videos, respectively? [6pts]**

**Sol:** Dimensional tensors used are:

* Time Series -> 3D tensors
* Images -> 4D tensors
* Videos -> 5D tensors

1. **In Pytorch, what method(s) is used to move tensors between CPU and GPU? [2pts]**

**Sol:** Methods is used to move tensors between CPU and GPU are:

* Every Tensor in PyTorch has a to() member function. Its job is to put the tensor on which it's called to a certain device whether it be the CPU or a certain GPU. We can also move a tensor to a certain GPU by giving its index as the argument to to() function.
* Another way to put tensors on GPUs is to call cuda(n) function on them where n is the index of the GPU. If you just call cuda, then the tensor is placed on GPU 0. The torch.nn.Module class also has to() and cuda() functions which puts the entire network on a particular device.
* Use nn.parallel.DistibutedDataParallel instead of multiprocessing or nn.DataParallel.
* There are two ways how we could make use of multiple GPUs: Data Parallelism and Model Parallelism.

1. **If we want to store 64 RGB images of 128x128 pixels, write this tensor. [2pts]**

**Sol: [**64 \*3 \* 128 \* 128]

1. **What type(s) of neural networks are good to use data parallelism? [4pts]**

**Sol:** The Neural networks that are wins for Data parallelism are:

1. Convolutional Neural Networks

Suitable for spatial data such as images.

This network takes fixed size inputs and generates fixed size outputs.

Ideal for images and video processing.

1. Recurrent Neural Networks

Suitable for temporal data, also called sequential data.

Can handle arbitrary input/output lengths.

Ideal for text and speech analysis.

1. **Under what circumstances can we develop model parallelism for deep learning training on multiple GPUs? [4pts]**

**Sol:** Model Parallelism can be uses:

* + - * The number of parameters of the model is large.
      * The number of layers in the model is large.
      * Model cannot fit a Single GPU.
      * Using Model Parallelism on Multiple GPU.

1. **What is profiling? Why should I profile my code? [6pts]**

**Sol:** Profiling refers to measuring the space (memory) or time complexity of a program, the

usage of particular instructions, or the frequency and duration of function calls. In the era of GPU-accelerated deep learning, when profiling deep neural networks, it is important to understand CPU, GPU, and even memory bottlenecks, which could cause slowdowns in training or inference.

In the deep learning field, the programs are not directly written by hand, most models are

trained and executed inside a framework. Therefore, a low-level profiling would be

useless, since it is not easily possible to change the way the framework executes our code.

By profiling the code, you can understand if there are any bottlenecks or memory leaks.

We can profile our training code using :

* + Nvidia-smi
  + PyTorch and PyProf
  + NVIDIA Nsight Systems

*Why should we profile our code?*

* Obtaining profiling information is a critical step before attempting to optimize code, as it enables you to focus your efforts on improving the parts of the code that will result in the biggest gains in performance.
* Help identify performance problems without having to touch their code.
* Aggregates Kernel performance.

1. **How many types of precision mixed calculations does the GPU provide? [3pts]**

**Sol:**

* 8-16x speedup for math-bound operations with TC vs FP32.
* 2x speedup for memory-bound operations vs FP32
* Fp16 reduces storage requirements for activation and weight tensors.

1. **Using AMP in Pytorch, what are the roles of the GradScaler and autocast? [4pts]**

**Sol: GradScaler**

* Gradient scaling helps prevent gradients with small magnitudes from flushing to zero (“underflowing”) when training with mixed precision.
* GradScalar starts with a small loss multiplier, which every so often it doubles. This gradual doubling behavior continues until GradScalar encounters a gradient update containing inf values. GradScalar discards this batch (e.g., the gradient update is skipped), halves the loss multiplier, and resets its doubling cooldown.
* GradScalar needs to exert control over the gradient update calculations (to check for overflow) and over the optimizer (to turn discarded batches into a no-op) to implement its behavior.
* GradScalar is a stateful object.
* torch.cuda.amp.GradScaler performs the steps of gradient scaling conveniently.

**Autocast**

* Instances of torch.cuda.amp.autocast serve as context managers that allow regions of your script to run in mixed precision.
* In these regions, CUDA ops run in a dtype chosen by autocast to improve performance while maintaining accuracy
* When entering an autocast-enabled region, Tensors may be any type. You should not call .half() on your model(s) or inputs when using autocasting.
* autocast should wrap only the forward pass(es) of your network, including the loss computation(s). Backward passes under autocast are not recommended. Backward ops run in the same type that autocast used for corresponding forward ops.

PART 2

Computation time on 1 GPU is: 14.069087505340576

Computation time on 2 GPU is: 5.954516649246216