Report oft Practical Course on High-Performance Computing

Parallel Deep Learning pipelines using Go and MPI

Persentor: Silin Zhao Supervisor: Patrick Michaelis

September 27, 2022

Project notation

Youtube link

https://www.youtube.com/watch?v=2siZQBvRPuY&t=6s

Datasets

- This project source code can be found https://github.com/scofild429/go_mpi_network,This is the **README** page.
- Iris dataset (https://www.kaggle.com/datasets/saurabh00007/iriscsv)
- Intel image classification, (https://www.kaggle.com/datasets/puneet6060/intel-image-classification?resource=download). Download it, put archive it in the folder ./datasets/

All training data will equally divied for each training network, specially for mpi

Configuration example

- ./goai/.irisenv
- ./goai/.imgenv

```
inputdataDims=4
inputLayerNeurons=30
hiddenLayerNeurons=20
outputLayerNeurons=3
labelOnehotDims=3
numEpochs=100
learningRate=0.01
batchSize=4
```

Sumbit the job in cluster

no singularity, installing golang 1.18 was failed always
using binary executable code of golang, **go build**

```
#!/bin/bash
#SBATCH --job-name mpi-go-neural-network
#SBATCH -N 1
#SBATCH -p fat
#SBATCH -n 20
#SBATCH --time=01:30:00

module purge
module load openmpi

mpirun -n 20 ./goai
```

Deep learning's problem

As AI comes to deep learning, the computing resource becomes more critical for training process.

Applications:

- Image Classification
- NLP
- Semantic segmentation

Solution

- GPU
- TPU
- **Distributed learning**

Single network architecture

```
raining data -> inputLayer(w1, b1) -> dinputLayer
Normalization
dinputLayer -> hiddenLayer(w2, b2) -> dhiddenLayer
Normalization
dhiddenLayer -> OutputLayer(w3, b3) -> doutputLayer
```

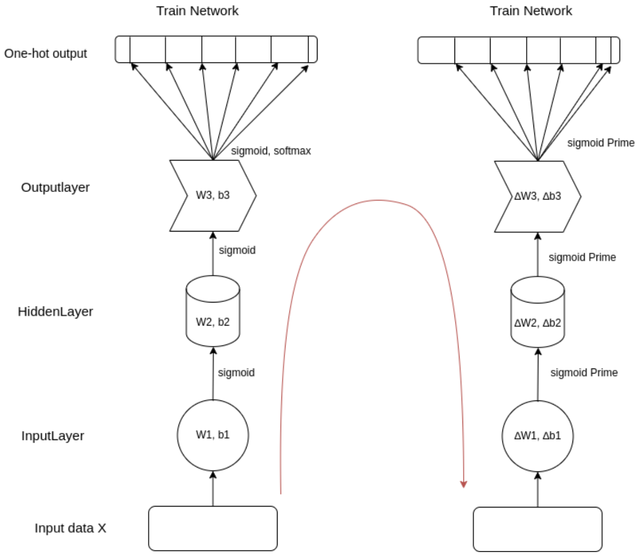
Loss $=$ L2: $(\text{doutputLayer - onehotlable})^2$

```
Backpropagation from Loss  of Outputlayer  to w3, b3
Backpropagation from error of Hiddenlayer  to w2, b2
Backpropagation from error of Inputlayer   to w1, b1
```

Derivative of sigmoid, Normalization, Standardization

- Stochastic Gradient Descent (SGD)
- Mini-batch Gradient Descent (MBGD)
- Batch Gradient Descent (BGD)

Illustration of weights updating



Code implementation

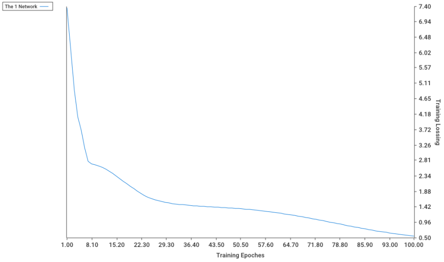
```
func main() {
^^Isinglenode.Single_node_iris(true)
^^Impicode.Mpi_iris_Allreduce()
^^Impicode.Mpi_iris_SendRecv()
^^Impicode.Mpi_images_Allreduce()
^^Impicode.Mpi_images_SendRecv()
}
```

You can review my code, and choose one of them to be executed in /goai/myai.go main function.
Comparing with python:

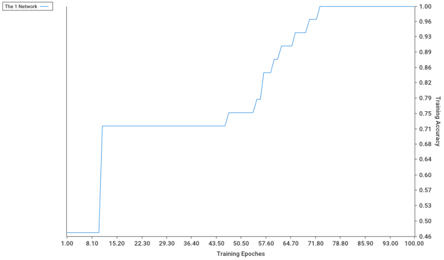
- ./pytorchDemo/irisfromscratch.py
- ./pytorchDemo/iriswithpytorch.py
- ./pytorchDemo/logisticRcuda.py

Network performance(iris dataset)

Loss



Accuarcy

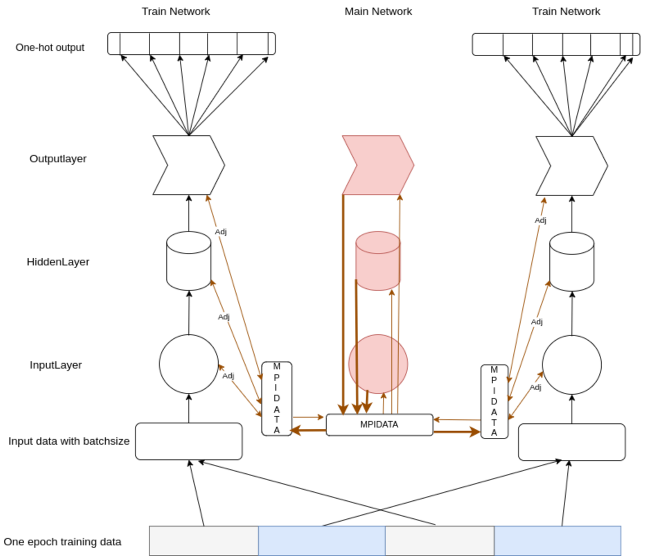


MPI communication

`github.com/sbromberger/gompi`
`import CGO as C`

- **Collective**
  - gompi.BcastFloat64s() -> C.MPI _Bcast()
  - gompi.AllreduceFloat64s -> C.MPI _Allreduce()
- **Non Collective**
  - gompi.SendFloat64s() -> C.MPI _Send()
  - gompi.SendFloat64() -> C.MPI _Send()
  - gompi.RecvFloat64s() -> C.MPI _Recv()
  - gompi.RecvFloat64() -> C.MPI _Recv()

Non collective architecture



Non collective design

rank $= 0$

- in **main network** weights will be initialized, but not for training,
- weights will broadcast to all other training networks

rank $!= 0$

- in **train network** receive weights from main network for initialization
- After each batch training done, sending its weights variance to main network

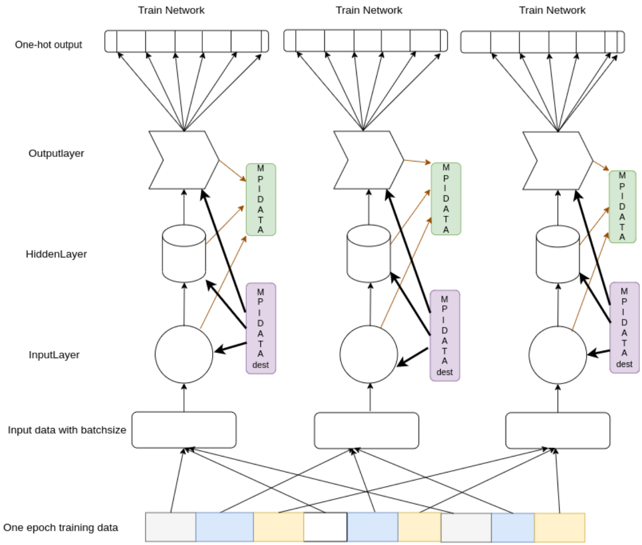
rank $= 0$

- receiving the variance from all training network
- accumulating and then sending back to training network

rank $!= 0$

- start next training batch

Collective architecture

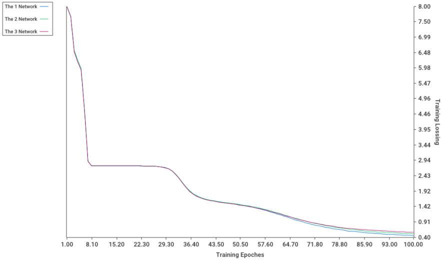


Collective design

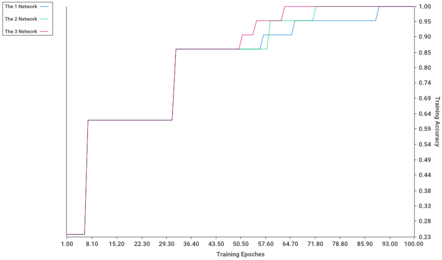
- All network train its data respectively,
- After each train batch, pack all weights into array
- $\mathrm{MPI}_{\mathrm{Allreduce}}$ for new array
- updating weights with new array

Iris dataset performance for non-collective

Send&Recv loss

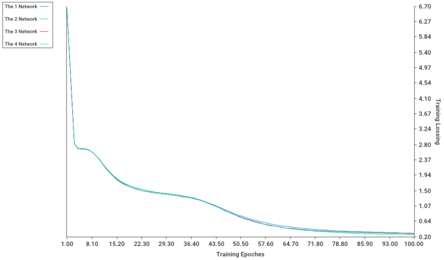


Send&Recv accuracy

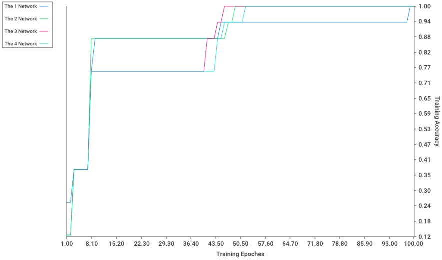


Iris dataset performance for collective

Allreduce loss

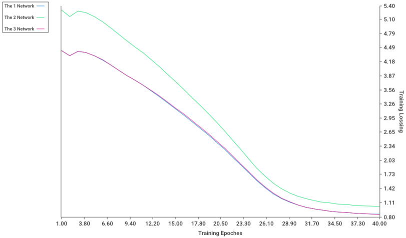


Allreduce accuracy

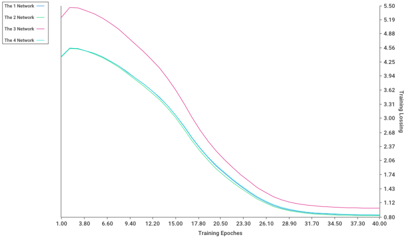


Intel image classification performance

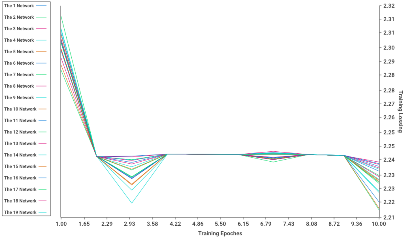
Send&Recv loss (220 images)



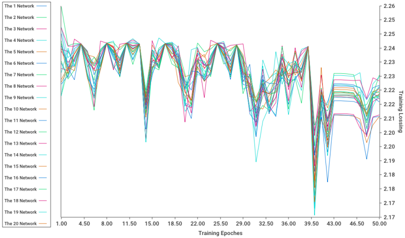
Allreduce loss (220 images)



SendRecv loss (14000 images)

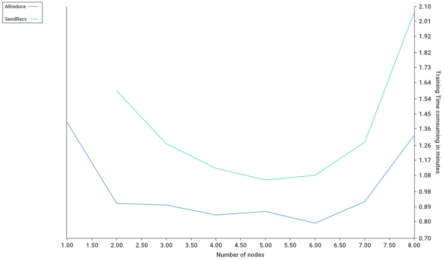


Allreduce loss (14000 images)

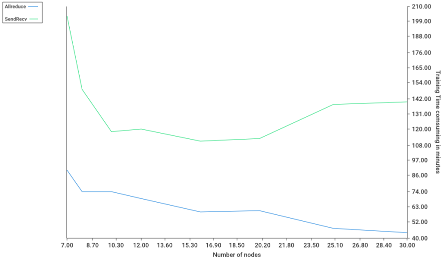


Speedup Diagrams

Iris for Allreduce and Send&Recv with different nodes



Intel Image Classification for Allreduce and Send&Recv with different nodes



Discussion

neural network model implement is not perfect, so the accuracy performance not so well

For each epoch:

- Allreduce: about 2 minutes
- Send&Recv: about 3.6 minutes, because of synchronization of each batch training

Change nodes, scaling behavior, such as speedup diagrams is missing

Change the batchsize, reducing mpi communication

Conclusion

- Golang can also be used for parallel computing
- neural network implementation of golang can be improved
- HPC cluster for distributed learning has significant benefits for large dataset