Analysis of Data Parallelism Acceleration on Speech Recognition and Synthesis ML Algorithms

Intern: Singyuan Yeh Advisor: Andrew Rosenberg, M.H. Chen, I.H. Chung and W.C. Wang

IBM TJ Watson Research Center, National Taiwan University

Abstract

The purpose is to accelerate the speech algorithm by parallelization using multiple GPUs. The analysis discusses some challenges and how they are addressed.

Introduction

This study accelerated the speech algorithm by data parallelism. The algorithm given by speech group is about converting the word to spectrum, including three models: (i) encoder, (ii) decoder and (iii) post process, as shown in Figure 1.

Figure 1: Flowchart of algorithm

The following is research method:

- **Data Parallelism:** Add data parallelism, maximize the GPUs utilization and find the bottleneck which reduced the utilization of GPUs.
- **Analysis:** Analyze each function and accelerate it further.

Parameter settings:

- batch size is 150 per GPU
- number of iteration is 180 per GPU

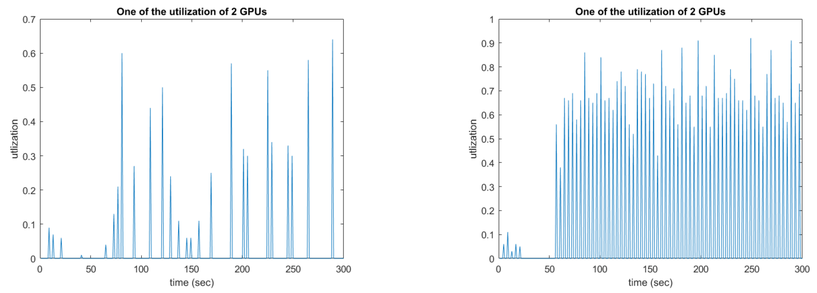
For example, if using 4 GPUs, we set the batch size as 600 and iteration time as 720.

Acknowledgment

I sincerely thank my advisor Prof. Rosenberg, Dr. Chung and Dr. Chen for the guidance and encouragement. Also, I would like to thank Prof. Wang for this intern opportunity at IBM Research. This project is partially supported by the Ministry of Science and Technology under Grant 107WFA0110351.

Performance Bottleneck

In order to accelerate the algorithm, the idle time of the GPUs has to be reduced. The function "trainloader" is to sort the data before generation. Figure 2 shows the relation between the utilization of the GPUs and the training time. The Subfigure (a) is sparser than the Subfigure (b). In other words, the function "trainloader" block the communication between CPU and GPUs.



(a) Utilization with function "trainloader" (b) Utilization without function "trainloader"

Figure 2: Comparison with and without function "trainloader"

Notes

The following experiment didn't use the function "trainloader".

First Few Steps

According to Figure 3, the warm-up steps should not be considered when doing performance evaluation.

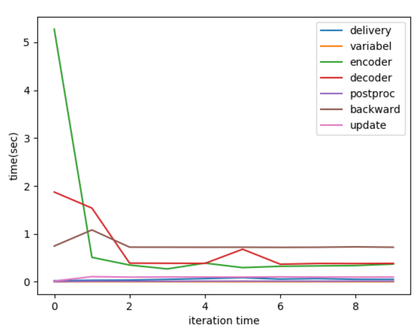


Figure 3: The time cost of first few step of each function with 8 GPUs

Performance Comparison

Figure 4 shows the relation between the GPU utilization and the training time taken in each function after the 4th iteration. The performance is improved, except for function "decoder" and "postproc".

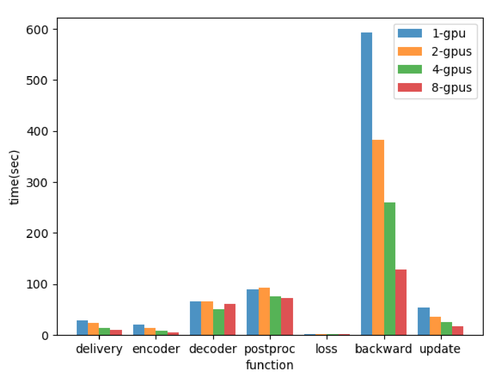


Figure 4: Comparison of the time of each function

Analyze the Model "Decoder" and "Postproc"

In order to know the performance of the model "decoder" and "postproc", the other two models are turned off and compute the time of the particular model, as shown in Figure 5.

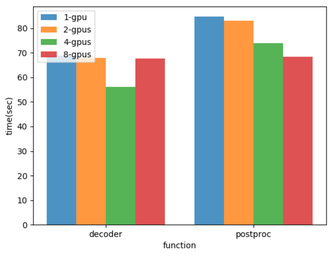


Figure 5: Time cost when running single module

According to Figure 5, The model "postproc" perform better when using more GPUs, but the model "decoder" is not.

Result

(i) The function "trainloader" is the bottleneck of the algorithm.

(ii) The module "encoder" performance is improved after adopting data parallelism.

Observation

(i) The first GPU used more memory than the other GPUs. It is possible that the performance will increase if balancing the memory usage between GPUs.

(ii) Adding parallelism to the "trainloader" function may also accelerate the algorithm.

(iii) The function "decoder" didn't perform better when using more GPUs. it may require further investigation.

Reference

[1] Andrew Rosenberg et al, *End-to-end Speech Recognition and keywords search on low resource language*