Projet - Rapport

IS324

CAYROL Pierre
FURELAUD Thomas

25 Janvier 2021

Le code en relation avec ce rapport se trouve ici :
https ://github.com/tfurelaud/CNN-Horovod

1 But du projet et état de l'art

Pour ce projet, nous avons décidé de travailler sur l'optimisation de performances d'un réseau de neurones de type *Convolutional Neural Network* (CNN). Pour ce faire, nous avions pour idée de distribuer le travail sur plusieurs GPU pour comparer les performances entre la version déjà implémentée que nous présenterons plus tard et cette nouvelle version. Nous avons donc découvert Horovod, qui permet de faire de distribuer l'étape d'apprentissage sur CPU et sur GPU [2]. Beaucoup d'implémentations de ce type existent, cependant, nous trouvions intéressant de procéder à ce travail par nous même, dans le but de mieux connaître cette façade du deep learning qui devient de plus en plus importante aujourd'hui.

Le but de ce projet a donc été dans un premier temps de se renseigner sur Horovod ainsi que sur son utilisation et son fonctionnement, pour ensuite, pouvoir distribuer le code de base.

2 Présentation d'Horovod

Comme nous le disions avant, Horovod est un framework développé par Uber, permettant de distribuer l'étape d'entraînement d'un code de deep learning qui utilise $TensorFlow$, $Keras$, $PyTorch$ et $MXNet$. Ce framework utilise le toolkit de NVIDIA : CUDA, $TensorFlow$ et OpenMPI, pour répartir la charge de travail de l'entraînement entre les GPU. Une des limites de ce framework est qu'il n'est pas possible de distribuer du travail sur d'autre GPU que des GPU NVIDIA.

Ce framework a donc été développé dans le but d'utiliser le *dataparallelism*. En effet, aujourd'hui la donnée a un effet d'entonnoir (*bottleneck*) dans beaucoup de processus d'entraînement et donc ralentit ce processus. Ainsi, comme les GPU modernes ont beaucoup de RAM, ils ont voulu développer ce framework dans l'optique d'étaler la donnée pour accélérer le processus d'entraînement. De plus, ils ont voulu améliorer la communication entre les GPU ainsi que l'étape du calcul de la moyenne des gradients, qui est une étape clé des algorithmes distribués de deep learning. Pour ce faire, ils ont implémenté un algorithme s'appuyant sur la technique *Ring AllReduce* [1] permettant d'améliorer ce point.

3 Implémentation

3.1 Code de base

Le code de base présent sur le GitHub donné en première page (sous le nom de $CNN.py$), est une implémentation d'un réseau de neurones de type CNN utilisant Keras. Il permet donc de faire de la classification d'images sur la base de donnée CIFAR-10. Il a été implémenté par nous même, l'an passé, dans le cadre d'une UE. C'est pourquoi nous trouvions intéressant d'optimiser ce code. L'architecture de ce réseau est assez basique, c'est pourquoi nous n'allons pas nous y attarder ici. Cependant, il est important

de préciser que ce code et son architecture ont été testés dans le but d'éviter le sur-apprentissage. De plus nous avons fixé le nombre d'*epoch* à 20 pour des raisons de visibilité sur les courbes présentant les résultats. Sinon un nombre d'*epoch* d'environ 15 est suffisant.

3.2 Utilisation d'Horovod

Voici le code rajouté pour l'utilisation d'Horovod ($CNN - Horovod.py$) :

```
import horovod.keras as hvd

hvd.init()

gpus = tf.config.experimental.list_physical_devices('GPU')
for gpu in gpus:
    tf.config.experimental.set_memory_growth(gpu, True)
if gpus:
    tf.config.experimental.set_visible_devices(gpus[hvd.local_rank()], '
    GPU')
```

Ci-dessus, nous initialisons la librairie et nous attribuons un GPU à chaque tâche.

```
    opt = tf.optimizers.Adam(0.001 * hvd.size())
    opt = hvd.DistributedOptimizer(opt)
    model.compile(loss= 'categorical_crossentropy',
                  optimizer = opt, metrics= ['accuracy'],
                  experimental_run_tf_function=False)
```

Ce code ci-dessus se trouve dans la construction du réseau. C'est ici qu'Horovod nous fait gagner des performances, en appelant l'*optimizer* distribué utilisant le système *Ring AllReduce* dont nous avons parlé plus tôt.

```
callbacks = [
    hvd.callbacks.BroadcastGlobalVariablesCallback(0),
]
```

Ce code est fait pour synchroniser l'état des différents GPU. Donc ici, le GPU 0 envoie ses informations à tout les autres GPU pour garantir l'initialisation cohérente de tous les GPU lorsque l'entraînement début avec des poids aléatoires ou des poids restauré à partir de *checkpoints*. Cette étape permet donc de s'assurer que tout les GPU débutent avec les mêmes poids. Dans le cas contraire, la moyenne du gradient ne pointera pas forcément vers un minimum.

```
if hvd.rank() == 0:
    callbacks.append(keras.callbacks.ModelCheckpoint('./checkpoint-{epoch
    }.h5'))
```

Et enfin, ce code a pour but de sauver l'état actuel de la session d'entraînement du GPU 0.

4 Comparaison

Ci dessous, nous comparons donc la version de base à la version TensorFlow-GPU et à Horovod. Pour ces expériences, nous avions 2 GPU à disposition qui sont 2 Tesla K40m. Nous n'avons, malheureusement, pas réussi à réaliser ces expériences sur plus de GPU. Nous nous attendons donc à avoir de meilleures performances que la version de base. Cependant, Horovod ne sera pas forcément meilleur que la version utilisant TensorFlow-GPU car comme on peut le voir sur les expériences de [2], la différence des performances est assez faible, même avec l'utilisation de 8 GPU. Ceci est normal, car l'un des points forts de Horovod est la communication des données entre les GPU, ainsi plus il y a de GPU, plus TensorFlow-GPU perd du temps, et plus Horovod devient performant.

Cependant, dans un premier temps nous devions nous assurer des performances de classification de notre modèle, pour être sûr qu'Horovod ou TensorFlow-GPU ne nous faisait pas perdre de précision, ou ne sur-apprenaient pas :

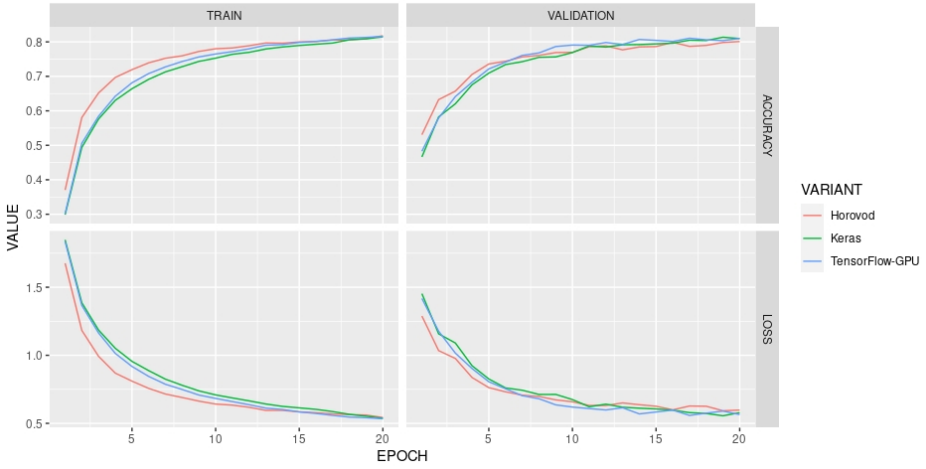

FIGURE 1 – Comparaisons des performances de classification des différentes variantes sur 20 époques. Ici TensorFlow-GPU et Horovod utilisent 2 GPU

On voit sur ces courbes ci-dessus, que ni TensorFlow-GPU ni Horovod ne sur-apprend et ne nous fait perdre en précision, ce qui est un point essentiel. Intéressons nous maintenant aux performances de ces 3 versions :

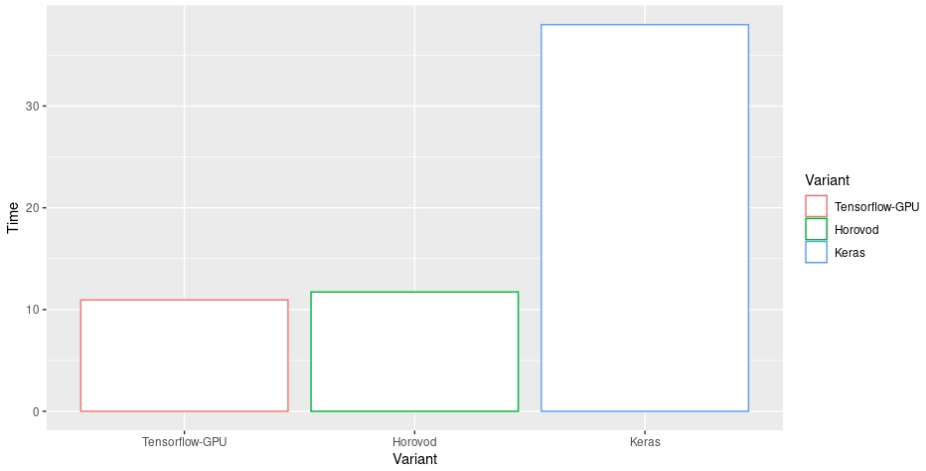

FIGURE 2 – Comparaison des performances des différentes variantes sur une époque. Ici TensorFlow-GPU et Horovod utilisent 2 GPU

On peut voir sur ce graphique que la distribution de l'étape d'entraînement sur des GPU nous fait gagner des performances, en passant de 39 secondes par époque sur le code de base à 12 avec l'utilisation de 2 GPU. Comme nous l'avions dit précédemment, Horovod ne se démarque pas de TensorFlow-GPU avec l'utilisation de 2 GPU. Il serait intéressant de faire ces tests avec plus de GPU par la suite, pour observer le comportement de notre implémentation. Pour finir, nous avons rapidement testé les performances en utilisant des CPU, mais nous ne le présentons pas ici, car ce n'est pas très intéressant niveau performance. En effet, les développeurs d'Horovod nous préviennent sur les mauvaises performances sur CPU (comparé à l'utilisation de GPU), ce qui s'est avéré correcte ici.

5 Difficultés rencontrées

Durant ce projet nous avons rencontré une difficulté majeure qui a été l'installation d'Horovod sur PlaFrim, car de nombreuses erreurs sont survenues au début. Ces erreurs étant assez mal expliquées, il nous a été très compliqué et long de comprendre la cause de ces erreurs qui été au final des conflits de versions des différents packages.

Références

[1] Pitch PATARASUK et X. YUAN. "Bandwidth optimal all-reduce algorithms for clusters of workstations". In : *J. Parallel Distributed Comput.* 69 (2009), p. 117-124.

[2] Alexander SERGEEV et Mike Del BALSO. "Horovod : fast and easy distributed deep learning in TensorFlow". In : *CoRR* abs/1802.05799 (2018). arXiv : 1802.05799. URL : http://arxiv.org/abs/1802.05799.