Programmation parallèle : Projet final

Jules Brablé
Yousra Lina El Khalloufi

ENSAE Paris
Year 2023-2024

Enseignants : X.Dupré & M.Durut

1 Introduction

En apprentissage machine et apprentissage profond, les modèles mobilisés on vu leur taille et performance croître de manière conséquente au cours de la dernière décennie. Très récemment, le champ de l'IA générative a vu naître une communauté particulièrement active, débouchant sur le développement de nombreux modèles *open-source*. Dans ce contexte, Meta a donné libre accès à sa gamme de modèles de type Llama (du 7B au 70B, en nombre de paramètres). Nous proposons ici de considérer de modèle Llama2-7B, et d'essayer d'améliorer son temps d'inférence en parallélisant en C++ une séquence d'opérations de ce modèle. En particulier, nous considérons le noeud encadré en vert ci-dessous :

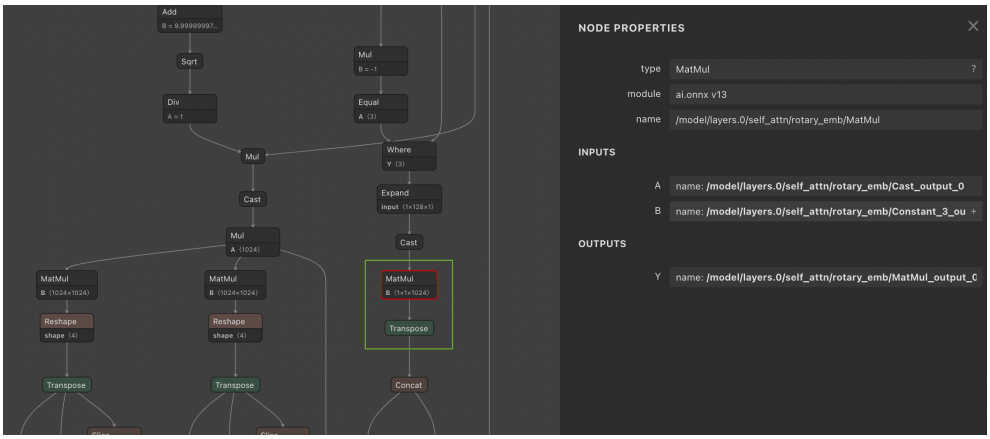

Figure 1 – Visualisation sous Netron du noeud à paralléliser

Il s'agit donc de paralléliser les opérations de multiplication de matrice et de transposition, originellement effectuées de manière séquentielle.

2 Expérimentation

2.1 Protocole expérimental

L'opération à réaliser est donc la suivante :

$$(A.B)^T$$

où Aet$B$ sont des matrices. Le noeud considéré prend comme entrées deux matrices de tailles différentes (1x1x1024) et (1x128x1). Et effectue probablement un broadcasting pour adapter les dimensions afin de réaliser la multiplication matricielle. Pour simplifier notre tâche, nous avons considérer que nos deux matrices d'entrées sont carrées et de mêmes dimensions.
Pour fusionner et optimiser les opérations séquentielles de multiplication et transposition, nous avons décidé de coder ces opérations en C++ puis de les englober avec Cython pour les rendre utilisables en Python. Pour analyser la performance, nous avons décidé de comparer entre :

- **C++** : code C++ pur (kernel_fusion.cpp), accompagné d'un code (interface.pyx) codé en Cython pour le passage en Python
- **Cython seul** : On code notre fonction $(A.B)^T$ directement en Cython (interface.pyx)
- **Numpy** : On code une fonction simple en Numpy. Numpy étant une bibliothèque de base de Python, il sied de l'inclure dans la comparaison.
- **TensorFlow** : La bibliothèque largement utilisée en Machine Learning, TensorFlow présente des performances marquantes en termes de calculs matriciels. Nous avons décidé de l'inclure dans nos comparaisons

— **PyTorch** : Cette bibliothèque permet de réaliser des calculs tensoriels rapides et précis. PyTorch est largement utilisé en Deep Learning.

2.2 Analyse des résultats

Nous utilisons le notebook (comparisons_notebook.ipynb) pour comparer et analyser la performance de ces différentes méthodes.
Nous commençons avec une première comparaison du temps d'exécution en fonction de la dimension de la matrice (dimensions allant de 1 jusqu'à $2^12 = 4096$ pour les 3 premières méthodes :

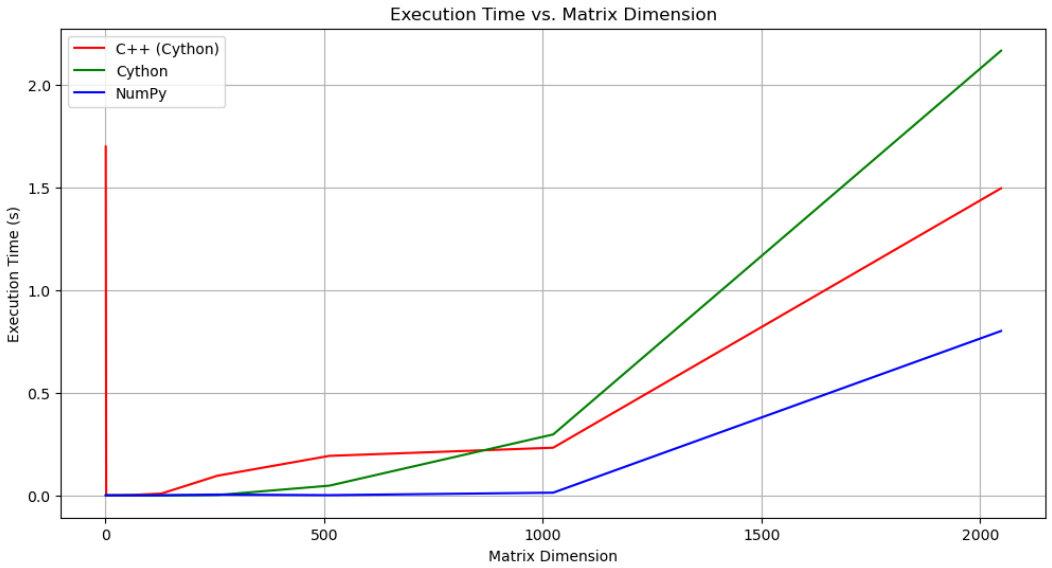

Figure 2 – Temps d'exécution en fonction de la dimension

Nous remarquons que C++ et Cython présentent des temps d'exécution plus important que Numpy. C'est de là que l'idée de comparer avec les autres frameworks a emmergé.
Nous avons alors fait la même comparaison mais en rajoutant TensorFlow et PyTorch.

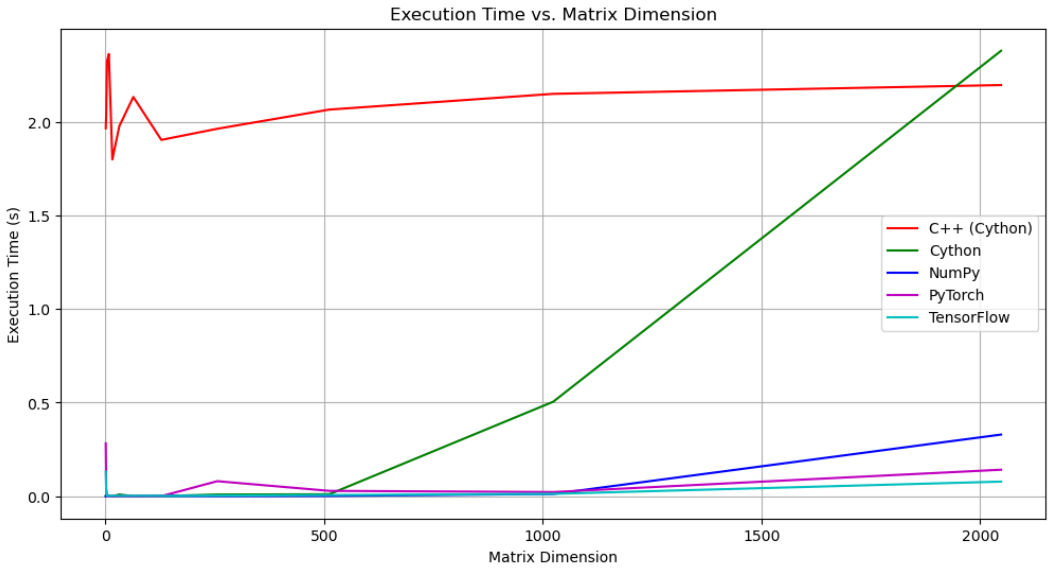

Figure 3 – Temps d'exécution en fonction de la dimension

La remarque demeure la même, C++ et Cython montrent des performances qui ne dépassent pas celles des bibilothèques pré-existantes de Python.

3 Conclusion et prolongements possibles

Pour conclure, si notre implémentation en parallèle des opérations matricielle ne permet pas de faire significativement mieux que les librairies natives, nous pouvons soulginer que ce projet reste ouvert à des améliorations. On pourrait par exemple continuer à améliorer notre code C++ pour améliorer le temps de calcul, par exemple en utilisant de la multiplication de matrices par blocs. Nous pouvons aussi étendre notre travail à des matrices de tailles différentes pour être dans un setting semblable au celui de Llama.