Scalabilité de l'Entraînement d'un Réseau de Convolution de Graphes pour l'Amarrage Moléculaire

Kévin Crampon[abc], Alexis Giorkallos[c], Stéphanie Baud[b], Luiz Angelo Steffenel[a]

[a]Université de Reims Champagne-Ardenne, LICIIS, LRC DIGIT, Reims, France

[b]Université de Reims Champagne-Ardenne, CNRS, MEDyC, Reims, France

[c]Atos SE, Center for Excellence in Advanced Computing, Echirolles, France

---

Résumé
L'apprentissage profond prend de plus en plus de place dans les simulations numériques, et la recherche de nouveaux médicaments ne fait pas exception. La recherche *in vitro* puis *in vivo* nécessite une première étape d'expérimentation *in sillico* : l'amarrage moléculaire (*molecular docking)*, et les années 2010 ont vu l'émergence d'un grand nombre de méthodes d'apprentissage profond appliquées à cette problématique. Aujourd'hui, deux grandes tendances apparaissent dans les modèles neuronaux pour l'amarrage moléculaire : l'augmentation de la taille des bases de données et l'utilisation des réseaux de convolutions reposant sur des graphes. Dans cet article, nous présentons une étude sur la scalabilité de l'entraînement d'un réseau de type convolution de graphes sur un nombre croissant de GPU. Nous montrons qu'il est possible d'obtenir une amélioration du temps d'entraînement jusqu'à 80%, à la dépense d'une dégradation des autres métriques d'évaluation du modèle.

Mots-clés : Scalabilité, Réseau de Convolution de Graphes, PyTorch, GPU

---

1. Introduction

La découverte de nouveaux traitements pharmaceutiques demande beaucoup de moyens techniques et financiers ; pour les réduire, de nombreuses méthodes de simulation numérique reposant sur des modèles sophistiqués et des optimisations sont désormais privilégiées. L'une des méthodes est la simulation d'amarrage moléculaire, utilisée par exemple pour la recherche de nouveaux principes actifs (ci-après appelés ligands) pouvant interagir avec une macromolécule biologique d'intérêt (nommée récepteur) telle qu'une protéine ; c'est cette dernière catégorie qui nous intéresse ici. Une telle interaction peut avoir plusieurs conséquences comme l'inhibition des effets de la macromolécule sur la santé du patient. Pour qu'il y ait interaction, le ligand doit se lier avec le récepteur : c'est alors qu'intervient la simulation d'amarrage moléculaire, qui permet de prédire, à partir de la structure 3D des deux molécules, la probabilité et e site de pose du ligand sur le récepteur.
L'objectif d'une simulation d'amarrage (présenté en Figure 1b) est de déterminer la conformation du complexe ligand-protéine, c'est-à-dire la position du ligand sur la surface de la protéine et la conformation de celui-ci, une conformation étant un arrangement possible des atomes d'une molécule respectant ses différents degrés de liberté. Une hypothèse simplificatrice couramment utilisée est de considérer la protéine comme rigide, c'est-à-dire, avec une

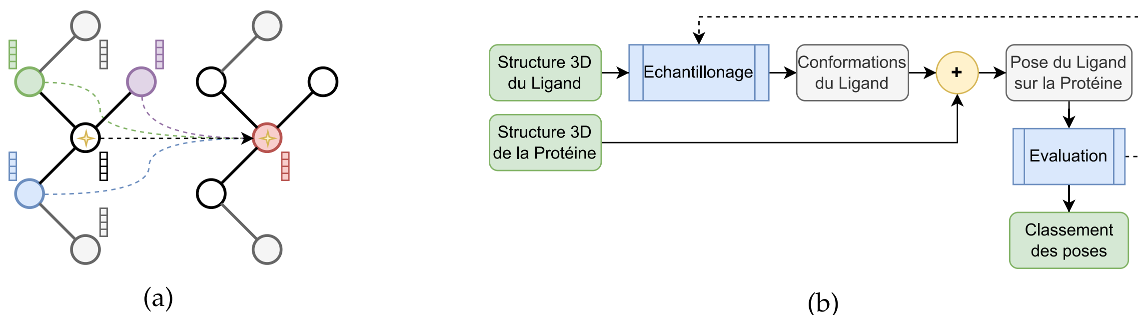


FIGURE 1 – (a) Une convolution spatiale sur un graphe, le nœud courant (ici identifié par une étoile) agrège l'information provenant de ses voisins directs (les nœuds vert, violet et bleu). Ainsi, le vecteur de caractéristiques du nœud courant est mis à jour comme le montre le graphe de droite. (b) Schéma simplifié du processus d'amarrage moléculaire, premièrement l'espace conformationnel du ligand est échantillonné explorant les différentes positions (à la surface de la protéine) et conformations possibles. Toutes les conformations générées sont ensuite associées à la protéine, ces poses sont ensuite évaluées par une fonction de score afin de les classer. De plus, certaines méthodes utilisent les scores produits afin de guider l'exploration de l'espace conformationnel suivante.

conformation fixe, ce qui permet de réduire le temps de calcul nécessaire à l'exploration de son espace conformationnel (dû à la taille importante de ce dernier).
Cet article se penche sur l'étude d'une fonction de score ou d'évaluation dont le but est de produire, à partir d'une pose du ligand sur le récepteur, un score qui reflétera au mieux l'affinité réelle de ce complexe. De nombreuses méthodes d'évaluation sans intelligence artificielle existent, notamment basées sur des calculs d'énergie, cependant nous nous intéressons à l'utilisation des réseaux de neurones à graphes pour résoudre ce problème.
En effet, depuis quelques années, la littérature sur l'apprentissage profond a migré vers les réseaux basés sur des graphes pour la représentation des données et les opérations de convolution [7]. Un graphe $\mathcal{G}(\mathcal{V}, \mathcal{E})$ est un ensemble de nœuds $\mathcal{V}$ liés par des arêtes de connectivité $\mathcal{E}$. Chaque nœud $v_i \in \mathcal{V}$ peut être agrémenté d'un ensemble de propriétés, notées $\overrightarrow{v_i}$. De même, chaque connexion peut porter un ensemble d'attributs $\overrightarrow{e_{i \to j}}$.
Une convolution graphe est une forme de généralisation d'une convolution sur maille cartésienne, où chaque nœud agrège l'information provenant de ses voisins ; le signal traverse la connexion qui les relie par un algorithme de message passing, figure (1a). L'information est alors pondérée par les poids entraînables du réseau (équation (1)).
L'avantage de représenter les données sous forme de graphes est que cette représentation reflète mieux la structure de l'objet. C'est notamment le cas des molécules dont les atomes sont traditionnellement discrétisés sur une grille 3D régulière où chaque atome prend les coordonnées de la plus proche cellule, ce qui représente une approximation du réel.

$$h_i^{(l+1)} = U_l(h_i^{(l)}, \sum_{j \in \mathcal{N}(i)} M_l(h_i^{(l)}, h_j^{(l)}, \overrightarrow{e_{j \to i}})) \quad (1)$$

Où $h_i^{(l)}$ est le vecteur de caractéristiques du nœud i à la couche l, $h_i^{(0)} = \overrightarrow{v_i}$, U_l et M_l sont des fonctions aux paramètres apprenables et $\mathcal{N}(i)$ est le voisinage du nœud i.

2. Données

Les données utilisées dans cette étude proviennent de la base de données PDBBind version 2020 [6]. Cette base de données contient entre autres 19 943 complexes ligand-protéine ainsi que les scores associés. Chaque complexe ligand-protéine est représenté par un graphe où les nœuds sont les atomes du complexe et des liens sont présents entre chaque couple de nœuds distant de moins de 2.5Å. Chaque nœud possède un ensemble de caractéristiques physico-chimiques détaillé en annexe A. Enfin, le voisinage de chaque nœud est encodé avec la méthode de *Random Walk* [3] d'ordre 20 de sorte que deux nœuds ayant un voisinage topologiquement identique aient le même encodage.

3. Architecture du réseau

3.1. GNN-LSPE

Dans cette étude, nous exploitons la méthode GNN-LSPE (*Graph Neural Networks with Learnable Structural and Positional Representations*) présentée par Dwivedi *et al.* [3]. Celle-ci sépare l'apprentissage sur les caractéristiques topologiques p des autres propriétés physico-chimiques x. En effet, ces auteurs proposent un modèle flexible, utilisable avec la plupart des couches de convolution sur graphes. Les équations de mise à jour sont données par les relations (2) et (3).

$$h_i^{(l+1)} = f_x(h_i^{(l)} \| p_i^{(l)}) \tag{2}$$

$$p_i^{(l+1)} = f_p(p_i^{(l)}) \tag{3}$$

Avec $\|$ qui est l'opérateur de concaténation et les fonctions f_x et f_p qui sont des opérations de convolution sur des graphes.

3.2. Architecture générale

Dans cet article nous utilisons la couche de convolution *Graph Attentional* (GAT) [2] à la fois pour f_x et f_p, dont l'équation de mise à jour des nœuds est présentée en Annexe B.
Le réseau consiste en une succession de 8 couches de type GAT-LSPE chacune de taille 64, puis chaque nœud est donné en entrée à un Perceptron Multi-Couches (MLP) de 3 couches de tailles décroissantes, donnant un degré de liberté du modèle à 94 000 paramètres. La dernière couche assure la régression sur chaque noeud, puis la moyenne de ces valeurs est prise comme score du complexe. Le détail de l'architecture du réseau utilisé est donnée en annexe C.
Nous avons implémenté la méthode GNN-LSPE avec le framework PyTorch-Geometric [4] et l'ensemble de notre code intègre la logique et les facilités de PyTorch Lightning.

4. Présentation des expérimentations

L'objectif de cette étude est de mesurer la scalabilité de l'entraînement d'un réseau de type convolution sur des graphes (GCN) sur plusieurs GPU. En effet, le framework PyTorch Lightning permet de gérer facilement l'utilisation d'un ou plusieurs GPU pour l'entraînement. Ainsi, nous avons testé la scalabilité sur 1, 2, 4 et 8 GPU, en comparant les temps d'entraînement sur 80 époques ainsi que la précision des métriques sur ces scenarii. Nous nous intéressons à deux métriques : le Coefficient de Détermination R^2 (Eq 4) et le Coefficient de Corrélation de Pearson R_p (Eq 5), ces deux métriques ayant une valeur optimale de 1.0. Les temps présentés dans la suite sont les temps moyens sur 10 expérimentations.

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - x_i)^2}{\sum_{i=1}^{n}(y_i - \overline{y})^2} \tag{4}$$

$$R_p = \frac{\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})}{\sum_{i=1}^{n}(x_i - \overline{x})^2 \sum_{i=1}^{n}(y_i - \overline{y})^2} \tag{5}$$

avec x_i la valeur prédite pour l'élément $i \in [\![1;n]\!]$, y_i la valeur cible, $\overline{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$ et $\overline{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$.

4.1. Méthode de distribution

Nous utilisons la méthode de distribution de base de PyTorch Lightning : *Distributed Data Parallel* [5], où chaque GPU possède son propre processus et ne voit qu'un sous-ensemble du dataset. Ce sous-ensemble est aléatoirement généré et demeure le même tout au long de l'entraînement, de plus, chaque graphe n'appartient qu'à un seul sous-ensemble. Enfin, tous les processus initialisent le modèle (les poids initiaux sont identiques sur tous les modèles) et procèdent à une propagation en avant et une rétro-propagation (afin de calculer les gradients) en parallèle. Enfin, les gradients sont collectés entre les processus puis moyennés pour que chaque optimiseur puisse mettre à jour ses poids et le modèle. Un schéma est présenté en Annexe E. Ainsi, à tout instant, les différents modèles ont des poids identiques.

4.2. Matériel

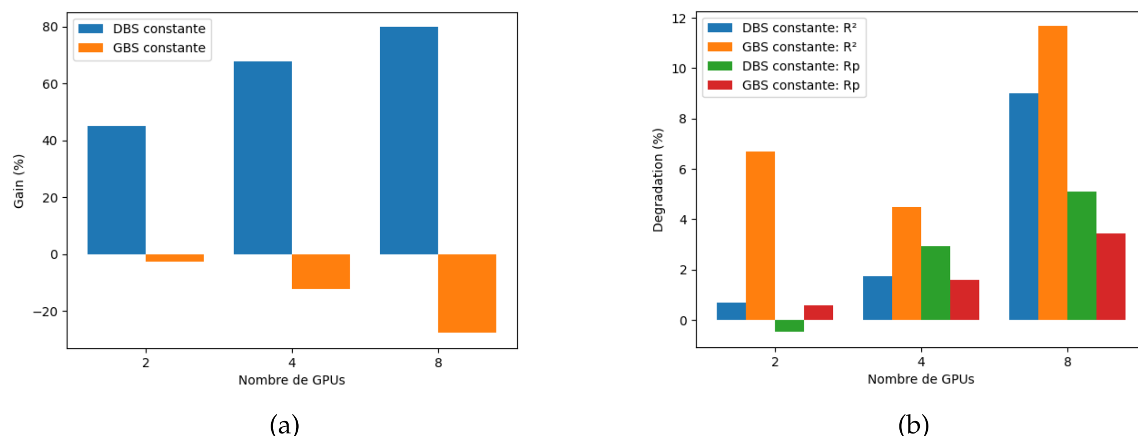
Les différentes expérimentations ont été réalisées sur un serveur Nvidia DGX-1 hébergé par le centre de Calcul Régional ROMEO de l'Université de Reims Champagne-Ardenne. Ce serveur est équipé de 8 GPU Nvidia Tesla V100 avec 16GB de VRAM chacun, dispose de deux sockets de 20 cœurs chacun et 2 threads par cœur, et possède 512GB de RAM. Un schéma du DGX est présenté en Annexe D.

5. Résultats

L'objectif ici, est de comparer deux méthodes de distributions. La première maintient une taille de batch distribué sur chaque GPU constante (la *device batch size* DBS), ce qui implique que la taille du batch global GBS ($\mathrm{GBS} = \#\mathrm{GPU} \times \mathrm{DBS}$) augmente avec le nombre de GPU utilisé. La seconde méthode maintient la GBS constante, ainsi la DBS diminue alors que le nombre de GPU augmente.
Les figures (2) et (3) montrent les performances pour une taille de batch respectivement de 64 et 128 appliquées aux deux stratégies (DBS constante et GBS constante) avec 2, 4 et 8 GPU, comparées à l'utilisation d'un seul GPU, sachant que pour 1 GPU les deux stratégies sont équivalentes, pour une taille de batch donnée.

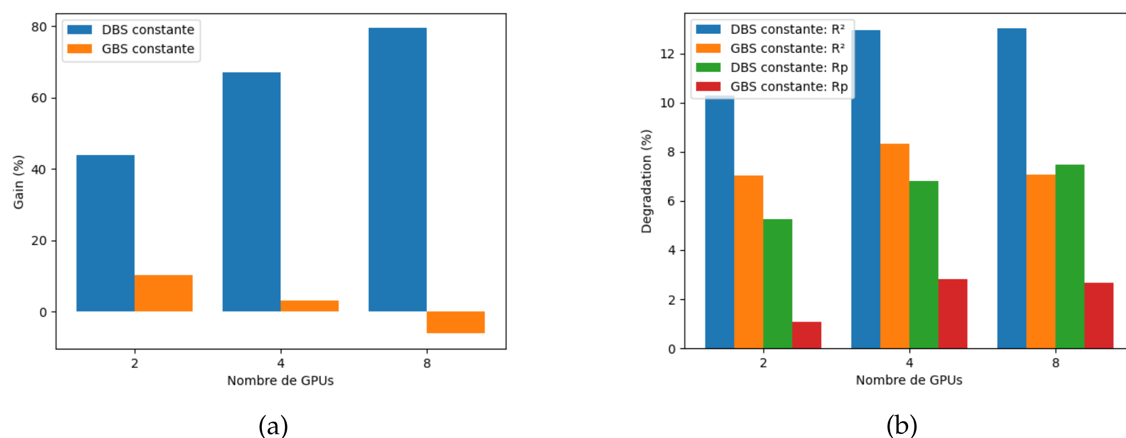
5.1. Comparaison des deux stratégies

Quelle que soit la taille de batch étudiée, la stratégie d'une DBS constante apporte un gain important en temps d'entraînement qui croit avec le nombre de GPU atteignant ainsi les 80% de réduction. La stratégie d'une GBS constante elle, augmente (ou dans un cas n'améliore que légèrement) ce temps. En ce qui concerne les métriques métiers, les deux stratégies provoquent une détérioration des scores, dans certains cas, cette détérioration est légère, mais le plus souvent elle est assez conséquente dépassant le plus souvent les 5%.



(a) (b)

FIGURE 2 – Taille du batch de 64 appliquée aux deux stratégies. (a) Gain de temps (en %) par rapport à l'entraînement de base sur 1 GPU. (b) Perte de précision sur les métriques avec l'augmentation du nombre de GPU.



(a) (b)

FIGURE 3 – Taille du batch de 128 appliquée aux deux stratégies. (a) Gain de temps (en %) par rapport à l'entraînement de base sur 1 GPU. (b) Perte de précision sur les métriques avec l'augmentation du nombre de GPU.

5.2. Étude de l'impact de la taille du batch

Lorsque la stratégie d'une GBS constante est utilisée, les DBSs sont alors d'autant plus petites que le nombre de GPU utilisé est grand. Ainsi l'utilisation de plusieurs GPU avec de petits batchs provoque un accroissement du nombre de communications nécessaires (copie du sous-dataset du CPU vers le GPU, mise en commun des gradients entre les GPU, copie des résultats du GPU sur le CPU) et donc cela met bien en évidence que les communications sont le goulot d'étranglement ici. Ainsi, l'utilisation de plusieurs GPU n'a un intérêt qu'avec une DBS assez grande.

Enfin l'annexe F qui présente les temps moyens d'entraînement pour chaque stratégie et chaque taille permet de voir que bien que la dégradation des métriques métiers augmente avec le nombre de GPU (pour les deux stratégies), un compromis peut être trouvé afin de tout de même profiter de l'accélération substantielle que fournit l'utilisation de plusieurs GPU. Dans

notre cas, nous pensons que le meilleur compromis est l'utilisation d'une DBS constante de 128.

5.3. Analyses

Évolution du temps d'entraînement

La perte de temps pour certaines expérimentations peut s'expliquer par le goulot d'étranglement que sont les communications aussi bien CPU-GPU qu'inter-GPU, cette perte de temps est d'autant plus importante que le nombre de GPU est grand et la DBS petite, cela s'explique en partie avec le fait que tous les GPU ne sont pas directement inter-connectés comme le montre le schéma du DGX-1 en annexe D. Ainsi, la mise en commun des gradients lors de l'étape de synchronisation avec 8 GPU demande de nombreuses communications qu'une petite taille de DBS (cas d'une GBS global de 64) ne permet pas de contrebalancer. La perte de temps peut aussi être causée par un déséquilibre entre les différents sous-datasets propres à chaque GPU, en effet la taille des graphes (nombre de nœuds et de liens) n'est pas uniforme, ainsi un sous-ensemble du dataset peut être plus gros et ainsi nécessiter plus de temps de calcul. Ce qui implique que les autres processus sont en attente pour la mise en commun des gradients.

Dégradation des métriques métier

Les figures (2b) et (3b) montrent une dégradation des métriques métiers avec l'utilisation de plusieurs GPU. Dans le cas de la stratégie de DBS constante (et donc une global batch size qui augmente avec le nombre de GPU), cette dégradation des métriques peut s'expliquer avec l'augmentation induite de la GBS. En effet, à chaque étape (consommation d'un batch entier) le réseau est mis à jour en fonction de l'erreur sur ce batch. Plus un batch contient d'éléments plus la probabilité que ceux-ci guident le réseau dans des directions opposées se renforce, ainsi les mises à jour ne permettent plus d'entraîner le réseau vers de meilleures performances et *in fine*, il stagne.

Dans le cas des deux stratégies, la dégradation des métriques métiers peut s'expliquer par les différentes moyennes effectuées qui peuvent réduire l'influence des mises à jour et ainsi ralentir la convergence du réseau.

6. Conclusion

Cette étude a pour but de déterminer la meilleure stratégie de parallélisation pour les futures campagnes d'entraînement, dans le cadre de nos recherches. Au regard des résultats obtenus, l'utilisation de 4 GPU avec une taille de batch de 64 par GPU et une stratégie de DBS constante, semble donner le meilleur rapport gain de temps par rapport à la détérioration des métriques sur ce cas d'usage précis. Cependant, il sera intéressant d'étudier ces compromis pour d'autres tailles, ainsi que pour d'autres pas d'apprentissage. Un pas d'apprentissage plus petit, permettant une convergence plus lente et plus stable, pourrait permettre de réduire les écarts des métriques en fonctions du nombre de GPU, au risque d'allonger le temps d'entraînement nécessaire à l'obtention de valeurs de même ordre. De plus, une étude approfondie est nécessaire pour comprendre pourquoi à GBS constante les métriques métiers se dégradent.

Bibliographie

1. Bernaschi (M.), Agostini (E.) et Rossetti (D.). – Benchmarking multi-gpu applications on modern multi-gpu integrated systems. *Concurrency and Computation : Practice and Experience*, vol. 33, n14, 2021, p. e5470. – e5470 cpe.5470.
2. Brody (S.), Alon (U.) et Yahav (E.). – How attentive are graph attention networks ?, 2021.
3. Dwivedi (V. P.), Luu (A. T.), Laurent (T.), Bengio (Y.) et Bresson (X.). – Graph neural networks with learnable structural and positional representations. – In *International Conference on Learning Representations*, 2022.
4. Fey (M.) et Lenssen (J. E.). – Fast graph representation learning with PyTorch Geometric. – In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.
5. Li (S.), Zhao (Y.), Varma (R.), Salpekar (O.), Noordhuis (P.), Li (T.), Paszke (A.), Smith (J.), Vaughan (B.), Damania (P.) et Chintala (S.). – Pytorch distributed : experiences on accelerating data parallel training. *Proceedings of the VLDB Endowment*, vol. 13, n12, Aug 2020, p. 3005–3018.
6. Liu (Z.), Su (M.), Han (L.), Liu (J.), Yang (Q.), Li (Y.) et Wang (R.). – Forging the basis for developing protein–ligand interaction scoring functions. *Accounts of chemical research*, vol. 50, n2, 2017, pp. 302–309.
7. Zhou (J.), Cui (G.), Zhang (Z.), Yang (C.), Liu (Z.), Wang (L.), Li (C.) et Sun (M.). – Graph neural networks : A review of methods and applications. *arXiv :1812.08434 [cs, stat]*, Jul 2019. – arXiv : 1812.08434.

Annexes

A. Caractéristiques physico-chimoques utilisées comme descripteurs des nœuds

Descripteur	Taille	Description
Type	8	Variable catégorielle (B, C, N, O, F, P, S, Others)
Hybridation	7	Variable catégorielle (Other, sp, sp2, sp3, sq. planar, trig. bipy, octahedral)
Degré en atome lourd	7	Variable catégorielle (0, 1, 2, 3, 4, 5, 6+)
Degré en hétéro-atome	7	Variable catégorielle (0, 1, 2, 3, 4, 5, 6+)
Charge Partielle	1	Float
Est hydrophobe	1	Booléen
Est dans un cycle aromatique	1	Booléen
Est accepteur d'hydrogène	1	Booléen
Est donneur d'hydrogène	1	Booléen
Est dans un cycle	1	Booléen

B. Couche de convolution de type GAT

L'équation de mise à jour des nœuds est présentée Eq (8).

$$e(h_i, h_j) = \text{LeakyReLU}(a^\top W h_i || W h_j) \tag{6}$$

Où la fonction e permet d'évaluer le lien allant du nœud j au nœud i, a et W étant apprenables.

$$\alpha_{ij} = \text{softmax}_j(e(h_i, h_j)) = \frac{\exp(e(h_i, h_j))}{\sum_{j' \in \mathcal{N}_i} \exp(e(h_i, h_{j'}))} \tag{7}$$

Comme expliqué par les auteurs [2], tous les coefficients sont normalisés par une fonction $softmax$ appliquée à tous les voisins d'un nœud.

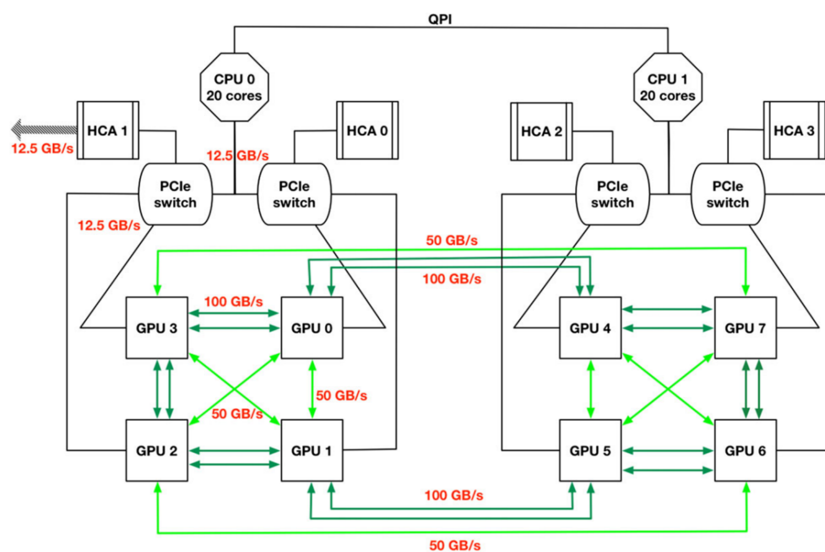
$$h_i^{(l+1)} = \sigma(\sum_{j \in \mathcal{N}(i)} \alpha_{ij} W h_j) \tag{8}$$

Avec σ une fonction non-linéaire.

C. Architecture détaillée du réseau

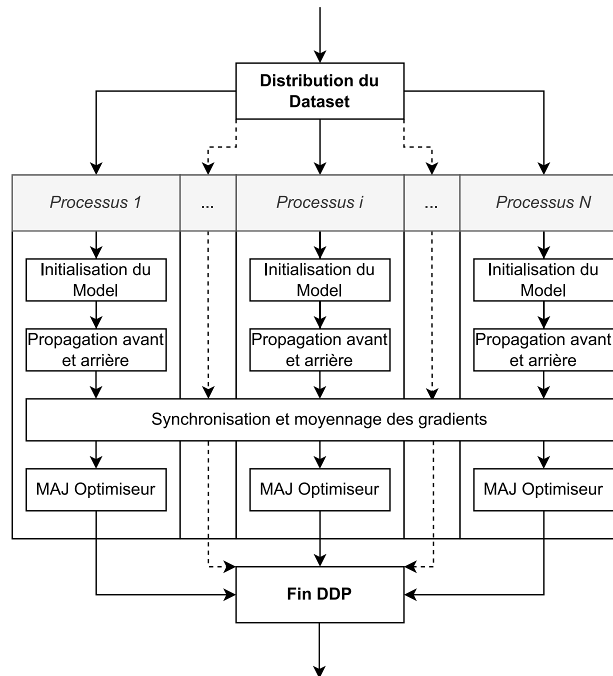
Paramètre	**Valeur**
GAT-LSPE Nombre de couches	8
GAT-LSPE f_x tailles	64
GAT-LSPE f_p tailles	64
GAT-LSPE Fonction d'activation	ReLU
MLP taille des couches	[64, 32, 16, 1]
Learning rate	1e-3
Weight decay	1e-4
Dropout	0.25
Fonction de coût	MSE
Fonction de réduction des graphes	Mean
Normalisation sur Batch	$\surd$

D. Architecture d'un DGX-1 Tesla V100 [1]



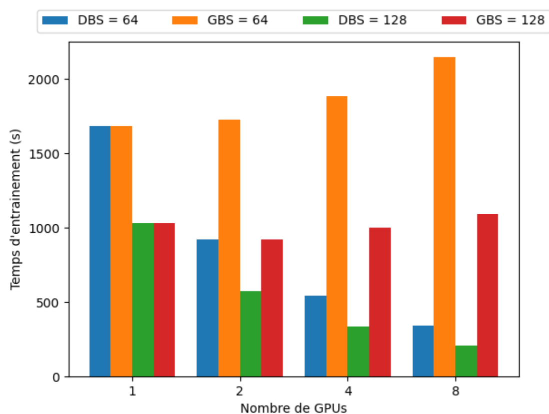
E. Méthode de distribution *Distributed Data Parallel* (DDP) de PyTorch-Lightning

La méthode DDP distribue le dataset sur les différents processus (1 processus par GPU), qui utilisent tous un modèle indépendant, mais avec des poids initiaux identiques, l'étape de mise en commun de gradient permet la synchronisation de ceux-ci en retournant la moyenne des gradients à tous les processus.

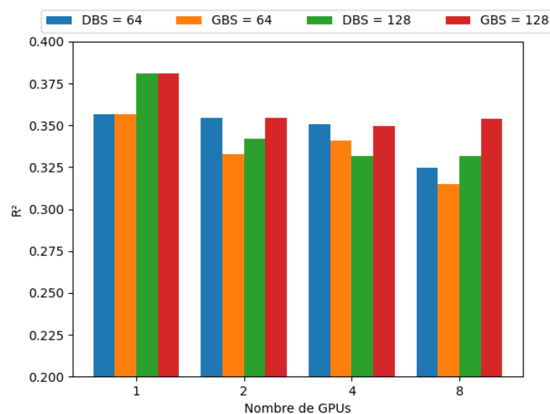


F. Temps et valeurs des métriques moyens des expérimentations effectuées

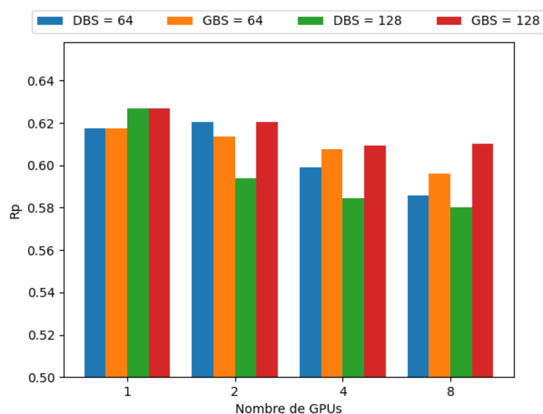
Valeurs moyennes des métriques utilisées lors des expérimentations. (a) Temps en seconde de l'entraînement, sur 80 epochs (b) Valeur du coefficient R^2. (c) Valeur du coefficient de corrélation de Pearson R_p.



(a)



(b)



(c)