TP BE MPI

Dion Thomas / Demarquet Alexandre

MODIA 5ème année
2025

Contents

List of Figures

1 Introduction

Ce BE a pour objectif d'implémenter et d'évaluer différentes stratégies de multiplication de matrices (GEMM – General Matrix-Matrix Multiplication) dans un environnement distribué, en utilisant la bibliothèque **MPI (Message Passing Interface)** et le simulateur **SimGrid**. L'approche se base sur une distribution des matrices selon un schéma **2D Block-Cyclic**, afin de répartir équitablement les calculs sur une grille de nœuds.

Trois variantes principales de l'algorithme GEMM seront implémentées :

- une version avec communications **bloquantes point à point**,
- une version utilisant les communications **collectives** via `MPI_Bcast`,
- une version optimisée avec **communications non bloquantes** permettant un recouvrement calcul/communication.

Pour évaluer les performances de ces 3 méthodes, un **benchmark** sera mené sur différentes tailles de matrices et configurations de grilles de nœuds. Les résultats seront écrits dans un fichier CSV et analysés à l'aide d'une **interface web**, écrite avec Streamlit. Cette interface permettra de visualiser clairement les écarts de performance entre les différentes stratégies, et facilitera l'analyse des traces SimGrid en fonction des différents paramètres.

2 Benchmark

2.1 Plan d'expérience

Pour évaluer les performances des différentes variantes de multiplication de matrices (GEMM), un script de benchmark a été mis en place `bench_final.sh`. Celui-ci repose sur l'exécutable `main` qui s'exécute dans un environnement simulé via **SimGrid**.

Le benchmark consiste à exécuter l'algorithme GEMM distribué en faisant varier plusieurs paramètres :

- la taille des matrices m, n et k, toujours égales pour simuler des matrices carrées, avec $n \in \{256, 512, 1024, 1536, 2048, 307$ (obtenu par $n = i \times b$, avec $i \in \{1, 2, 4, 6, 8, 12\}$ et $b = 256$) ;
- la taille des blocs locaux $b = 256$;
- la dimension de la grille logique $(p \times q)$, avec $p \in \{1, 2, 3, 4, 5, 6, 12, 16\}$ et $q \in \{1, 2, 3, 4, 5, 612, 16\}$, soit un nombre total de processus $P = p \times q$;
- l'algorithme utilisé : `p2p`, `bcast` ou `p2p-i-la` (point-à-point avec lookahead) ;
- le paramètre de `lookahead`, propre à la version `p2p-i-la`, avec une valeur variant de 1 à n/b.

Chaque combinaison de paramètres est exécutée plusieurs fois (5 itérations), et la performance est mesurée en nombre de **GFLOP/s**.

On repète ce benchmark sur différentes plateformes simgrid comme :

- `bin_tree256` (résultats : `results_bintree256.csv`)
- `cluster_crossbar` (résultats :`results_crossbar.csv`)
- `ring_256` (résultats : `results_ring256.csv`)

L'environnement est configuré pour utiliser que 1 thread pour OpenMP et MKL.

2.2 Interface web / visualisation des résultats

Pour faciliter la visualisation des résultats, nous avons développé un script utilisant **Streamlit** afin de créer une interface web interactive. Cela nous permet de générer facilement des *dashboards* pour analyser les performances (en GFLOPS) des différents algorithmes en fonction :

- du **nombre de processus** utilisés,
- de la **taille des matrices**,
- de la **répartition de la grille de processus** (grille carrée, plus de lignes que de colonnes, ou inversement),

- de la **plateforme SimGrid** sélectionnée pour l'exécution.

L'interface permet également d'étudier l'**influence du paramètre de lookahead** dans l'algorithme `P2P-i-la`, en observant son impact sur les performances.
Pour lancer l'interface graphique il suffit de télécharger streamlit (`pip install streamlit`) et de lancer l'interface avec la commande : `streamlit run st_benchmark.py`

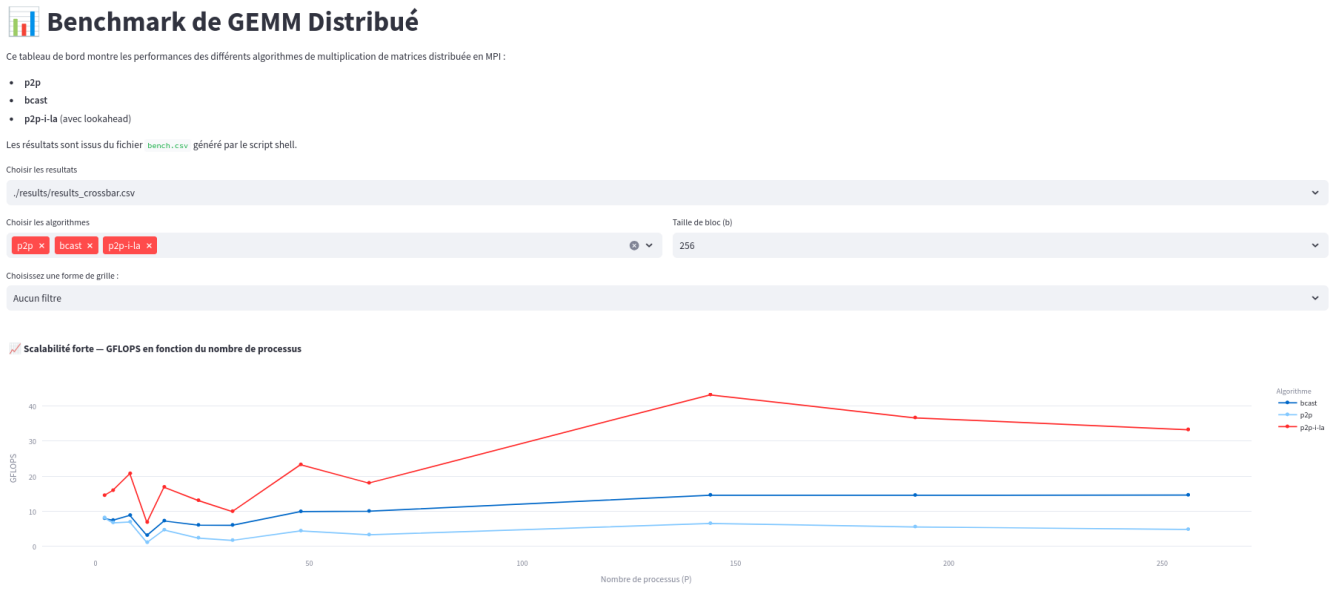

Figure 1: Screenshot de l'interface streamlit

2.3 Résultats

2.3.1 Analyse de la scalabilité forte

Pour évaluer la **scalabilité forte**, nous avons fixé la taille des matrices (par exemple $n = 2048$) et fait varier le **nombre de processus** utilisés pour l'exécution. Le but est d'observer comment les performances (en GFLOP/s) évoluent lorsque l'on augmente le parallélisme tout en gardant une charge de travail constante. Ici les résultats présentés ont été réalisé avec la plateforme `cluster_crossbar.xml`.

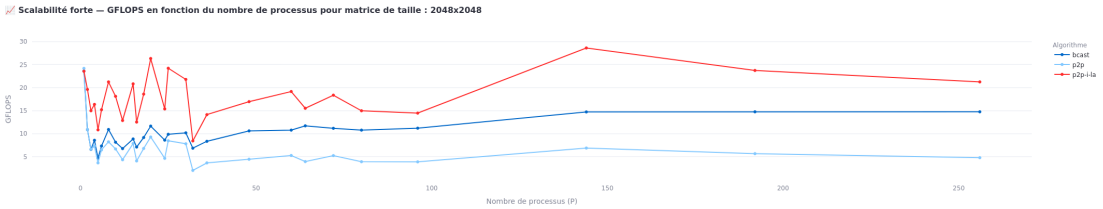

Figure 2: Scalabilité forte pour les trois variantes de GEMM avec $n = 2048$

- **Algorithme** `p2p-i-la` : C'est l'algorithme le plus performant globalement. Il atteint un pic de performance avoisinant les 27 GFLOPS autour de 144 processus. Toutefois, on observe une forte variabilité pour un faible nombre de processus. Au-delà de 144 processus, les performances restent élevées mais montrent une légère baisse.

- **Algorithme** `bcast` : Son comportement est plus stable. La performance augmente progressivement avec le nombre de processus, atteignant un plateau autour de 14 GFLOPS. Cela indique une scalabilité forte correcte jusqu'à une certaine limite, après quoi l'augmentation du nombre de processus n'apporte plus de gain significatif, probablement à cause des coûts de diffusion.

- **Algorithme** `p2p` : Il montre les performances les plus faibles. La montée en performance est très limitée (environ 8 à 9 GFLOPS au maximum), et on observe même une légère baisse au-delà de 64 processus. Cela indique que la communication point-à-point devient un goulot d'étranglement dans un contexte de scalabilité forte.

Ainsi l'algorithme `p2p-i-la` montre la meilleure scalabilité forte, bien qu'irrégulière pour un faible nombre de processus. L'algorithme `bcast` offre un bon compromis entre performance et stabilité. Enfin, `p2p` est clairement limité par la surcharge de communication, ce qui le rend inadapté pour des configurations massivement parallèles sur un problème de taille fixe.

2.3.2 Influence du lookahead et de la forme de la grille

Grâce à notre plateforme web interactive, nous avons pu explorer de manière dynamique les performances des différents algorithmes selon plusieurs paramètres, en particulier la valeur du *lookahead* et la forme de la grille de processus.

Effet du lookahead L'interactivité de l'interface nous a permis de visualiser directement les résultats pour plusieurs valeurs de *lookahead*. Il en ressort clairement que les plus grandes valeurs, notamment **7 et 8**, produisent les meilleures performances, et ce de manière significative. À l'inverse, les plus petites valeurs de *lookahead* (comme **1** et **2**) tendent à se rapprocher des performances de l'algorithme `bcast`.

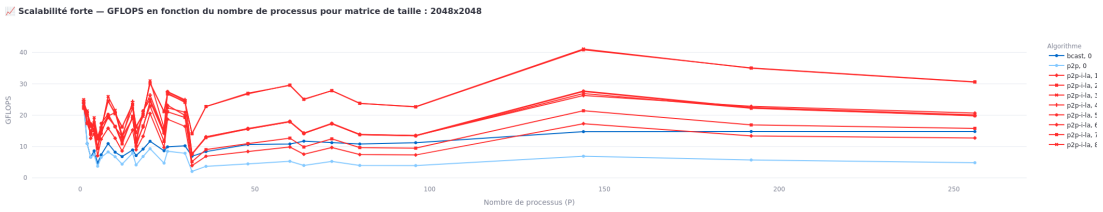

Figure 3: Comparaison des performances selon différentes valeurs de lookahead

Influence de la forme de la grille de processus Notre interface interactive permet également de visualiser les résultats en fonction de la forme de la grille de processus utilisée pour la distribution des tâches : $p = q$ (grille carrée), $p < q$ (grille étendue horizontalement), et $p > q$ (grille étendue verticalement).

Il est important de noter que les courbes présentées précédemment sont issues d'une **moyenne sur ces trois configurations**. En analysant séparément chaque cas, nous obtenons des comportements légèrement différents :

- **Grille carrée ($p = q$)** : Les performances sont presque constantes lorsque le nombre total de processus augmente (jusqu'à 256). On observe toujours la hiérarchie suivante : `p2p-i-la` `bcast` `p2p` , avec une décroissance nette de `p2p`. C'est dans cette configuration que l'on observe un pic de performance autour de 144 processus pour `p2p-i-la`.
- **Grille avec $p < q$** : Les résultats sont très similaires à la moyenne globale, sans le pic observé à 144 processus. Les performances restent cohérentes avec les tendances générales, mais sont un peu plus lissées.
- **Grille avec $p > q$** : Même comportement que pour $p < q$, avec des performances proches de la moyenne, et l'absence du pic de performance présent dans la grille carrée.

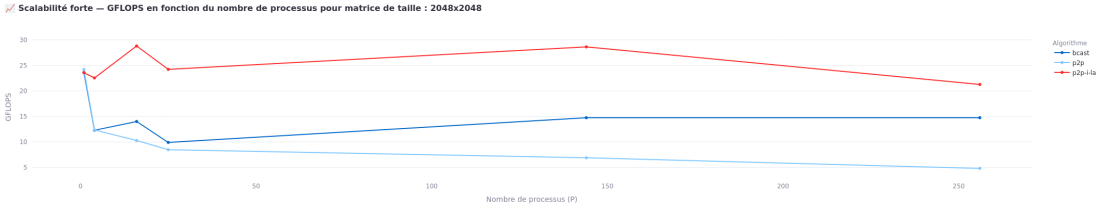

Figure 4: Performances pour une grille carrée ($p = q$)

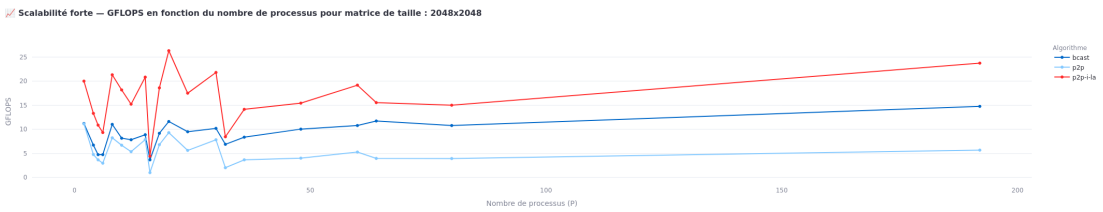

Figure 5: Performances pour une grille horizontale ($p < q$)

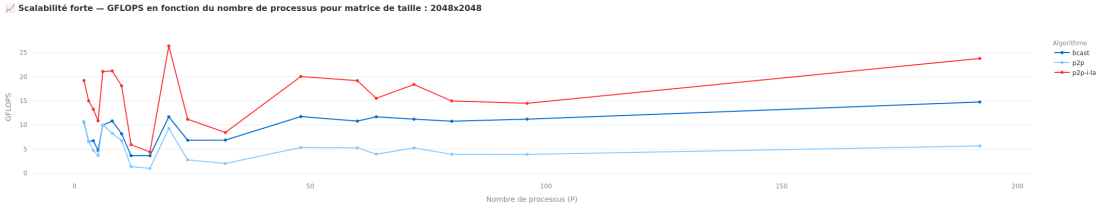

Figure 6: Performances pour une grille verticale ($p > q$)

Scalabilité forte — Matrice de taille 3072×3072

Nous avons également évalué la scalabilité forte sur une matrice de taille 3072×3072. Comme pour le cas précédent, la taille du problème est maintenue constante tandis que le nombre de processus varie. Les résultats obtenus sont présentés dans la figure suivante :

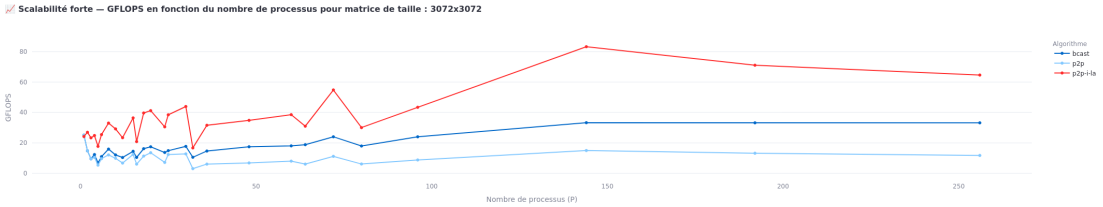

Figure 7: Scalabilité forte pour une matrice 3072×3072 — GFLOPS en fonction du nombre de processus

Les tendances générales observées précédemment se confirment ici. L'algorithme `p2p-i-a` domine largement, atteignant un pic de performance de près de **80 GFLOPS** autour de 144 processus. Même en dehors de ce pic, il maintient des performances élevées (entre **60 et 70 GFLOPS**).
Les algorithmes `bcast` et `p2p` plafonnent respectivement à environ **35 GFLOPS** et **15 GFLOPS**, avec peu d'amélioration au-delà d'un certain nombre de processus.

En explorant les différentes valeurs de *lookahead* via notre interface interactive, nous constatons à nouveau que ce sont les plus grandes valeurs (en particulier les **lookahead 11 et 12**) qui permettent d'obtenir les meilleures performances. Les plus petites valeurs de lookahead (1 à 4) produisent des résultats bien inférieurs, se rapprochant des performances de `bcast`, voire de `p2p`.

Ces résultats confirment qu'un lookahead élevé permet une anticipation plus efficace des communications et une meilleure exploitation du parallélisme.

2.3.3 Scalabilité faible — Influence de la taille de la matrice

Dans cette expérience, nous avons fixé le nombre de processus et fait varier la taille de la matrice n afin d'étudier la scalabilité faible des différents algorithmes. Les résultats sont présentés dans la figure suivante :

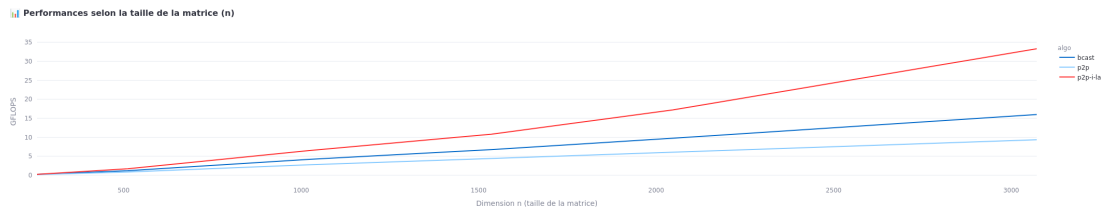

Figure 8: Performances (GFLOPS) en fonction de la taille de la matrice n

On observe que les trois algorithmes voient leurs performances augmenter régulièrement avec la taille du problème, comme attendu dans un contexte de scalabilité faible.

- L'algorithme `p2p-i-la` est de loin le plus performant, dépassant les **30 GFLOPS** pour $n = 3000$, avec une croissance quasi quadratique.
- L'algorithme `bcast` présente une croissance plus modérée, atteignant environ **16 GFLOPS** pour la plus grande taille.
- L'algorithme `p2p` est le moins efficace. Malgré une montée linéaire, il reste largement en dessous des deux autres, plafonnant à environ **9 GFLOPS**.

Ces résultats confirment que l'algorithme `p2p-i-la`, en anticipant les communications grâce à un lookahead bien choisi, est le plus adapté aux grands volumes de données. Il permet une meilleure exploitation des ressources, même avec un nombre de processus constant.

Scalabilité faible — Influence de la valeur du *lookahead* Impact de différentes valeurs de *lookahead* sur la scalabilité faible.

- Plus la taille de la matrice n augmente, plus il est bénéfique d'utiliser un *lookahead* élevé.
- Pour chaque taille de matrice testée, la meilleure performance est systématiquement obtenue avec la plus grande valeur de *lookahead* parmi celles évaluées.

Ce résultat montre que l'anticipation des communications devient de plus en plus cruciale à mesure que le volume de calcul augmente. Un grand *lookahead* permet de mieux exploiter la parallélisme tout en masquant les latences de communication.

La figure suivante illustre cette évolution des performances selon la taille de la matrice et la valeur de *lookahead* :

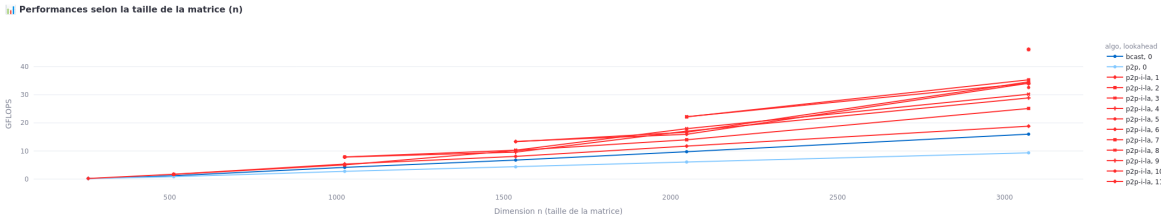

Figure 9: Performances (GFLOPS) en fonction de la taille n pour différentes valeurs de *lookahead* (scalabilité faible)

3 Conclusion générale

L'étude a permis de mettre en évidence les meilleures stratégies de communication pour des multiplications de matrices parallèles. En particulier, l'algorithme `p2p-i-la` combiné à une valeur de *lookahead* élevée fournit les meilleures performances, que ce soit en scalabilité forte ou faible.

Cependant, nous n'avons pas pu exploiter l'outil VITE comme prévu. Des difficultés techniques ont empêché la génération des traces d'exécution, ce qui nous a empêchés d'analyser finement l'utilisation du CPU, de la mémoire ou les synchronisations entre processus. Cette analyse qualitative serait un complément important à envisager dans la suite de ce travail.