# Projet de Programmation Numérique

Mohamed Anis KHODJA

Master 1 - Calcul Haute Performance, Simulation


Département Informatique
UVSQ - ISTY
France
Juillet 2021

**Introduction :**

Dans le cadre de notre Projet de Programmation Numérique, il nous est demandé de réaliser une bibliothèque de calcul matriciel dense en séquentielle et en parallèle pour architecture à mémoire distribuée, réalisant le produit scalaire de deux vecteurs réels, le produit matrice-vecteur (réels), et enfin le produit matrice-matrice (réels). Pour ce faire, nous allons :

- Faire une présentation des algorithmes et leur parallélisation.
- Une analyse de performance (Scalabilité forte et faible).

# Produit scalaire de deux vecteurs :

**Algorithme :**

$\underline{Début}$:

$Donnees$: $vect1 : vecteur\ [N]\ de\ reel$

$vect2 : vecteur\ [N] de\ reel$

$i : entier, variable\ indice\ de\ boucle$

$prod\_scalaire : réels, variable\ cumul\ resultat$

$prod\_scalaire = 0;$

$pour\ i\ de\ 0$ à $N - 1\ faire$

$prod\_scalaire\ = prod\_scalaire\ + vect1[i] * vect2[i];$

$Fin\ pour;$

$Retourner\ prod\_scalaire;$

$\underline{Fin}$;

L'algorithme est assez simple, le produit scalaire est obtenu en multipliant les éléments des deux vecteurs élément par élément, ensuite on fait la somme des résultats de multiplications, les deux vecteurs doive impérativement avoir la même taille, le résultat final est un réel qui représente le produit scalaire.

Pour la version parallèle, on va distribuer les éléments de chaque vecteur sur l'ensemble des processus grâce à la commande MPI_Scatter pour qu'ils fassent le calcule, le processus de rang 0 va envoyer les données correspondant aux deux vecteurs 1 et 2 aux autres processus pour faire le produit scalaire, chaque processus fera le produit case par case $vect1[i] * vect2[i]$ et calculera ensuite localement la somme des produits.

La commande MPI_Reduce permettra ensuite de faire la somme des valeurs enregistré dans chaque processus pour nous donner la somme global (résultats du produit scalaire) qui sera ensuite enregistré dans le processus 0.

| V1 | V2 |
|---|---|
| A1 | B1 |
| A2 | B2 |
| A3 | B3 |
| A4 | B4 |
| A5 | B5 |
| A6 | B6 |
| A7 | B7 |
| A8 | B8 |

Scatter

| Processus | 0 | 0 | 1 | 1 | 2 | 2 | 3 | 3 |
|---|---|---|---|---|---|---|---|---|
| V2 | A1 | A2 | A3 | A4 | A5 | A6 | A7 | A8 |
| V1 | B1 | B2 | B3 | B4 | B5 | B6 | B7 | B8 |

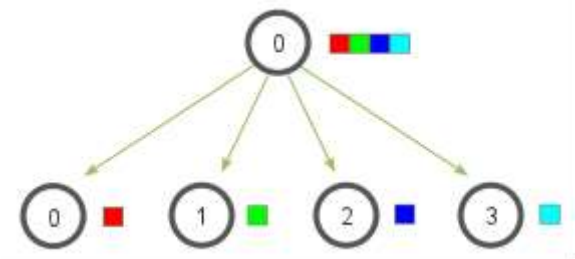
| Processus | | | | | | |
|---|---|---|---|---|---|---|
| 0 | A1*B1 | + | A2*B2 | = | X1 |
| 1 | A3*B3 | + | A4*B4 | = | X2 |
| 2 | A5*B5 | + | A6*B6 | = | X3 |
| 3 | A7*B7 | + | A8*B8 | = | X4 |

Reduce

| Processus | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0 | X1 | + | X2 | + | X3 | + | X4 | = | SOMME_TOTAL |

MPI_Scatter prend un tableau d'éléments et les distribue dans l'ordre sur les rangs de processus. Le nombre d'éléments envoyés à chaque processus (send_count) est donné en divisant la taille du tableau (taille du vecteur) sur le nombre de processus, si on a un vecteur de 4 éléments et on a 4 processus, alors le processus 0 obtient le premier élément du vecteur, le processus 1 obtient le deuxième, et ainsi de suite.

On prend l'exemple dans l'image ci-après, le premier élément (en rouge) va au processus 0, le deuxième élément (en vert) va au processus 1, et ainsi de suite. Bien que le processus racine (process zéro) contienne l'intégralité du tableau de données, MPI_Scatter copiera l'élément approprié dans le processus de réception.



Si on a un vecteur de taille 8 et toujours 4 processus, alors send_count vaut deux, alors le processus 0 obtient les premier et deuxième élément, le processus 1 obtient les troisième et quatrième, et ainsi de suite.

Cependant, il faut tenir compte du cas où la taille du vecteur et le nombre de processus ne sont pas divisible, c.à.d. que le reste de la division est diffèrent de zéro, dans ce cas on va envoyer les éléments restant du vecteur au dernier processus, par exemple si on a un vecteur de taille 6 et 4 processus, le processus 0 va prendre le premier élément, le processus 1 va prendre le deuxième, le processus 2 le troisième, et le dernier processus (processus 3) va prendre tous les éléments restant c.à.d. les éléments 4, 5 et 6 du vecteur.

**Algorithme parallèle :**

$Début$:

$Donnees$: $vect1 : vecteur\ [N]\ de\ reel$

$vect2 : vecteur\ [N] de\ reel$

$i : entier, variable\ indice\ de\ boucle$

$prod\_scalaire : réels, variable\ cumul\ resultat$

$prod\_scalaire = 0;$

$MPI\_Scatter\ (partagage\ vect1\ aux\ processus)$

$MPI\_Scatter\ (partagage\ vect2\ aux\ processus)$

$pour\ i\ de\ 0$ à $N - 1\ faire$

$prod\_scalaire\ = prod\_scalaire\ + vect1[i] * vect2[i];$

$Stockage$

$Fin\ pour;$

$MPI\_Reduce\ (envoi\ du\ résultat\ au\ processus\ zéro)$

$Affichage$

$Retourner\ prod\_scalaire;$

$Fin;$

# Produit matrice-vecteur :

**Algorithme :**

$\underline{Début}$:

$Donnees$: $vect : vecteur\ [N]\ de\ reel$

$mat : matrice\ [M][N]\ de\ reel$

$i, j : entier, variable\ indice\ de\ boucle$

$prod : vecteur\ [M]\ de\ reel, resultat$

$pour\ i\ de\ 0$ à $M - 1\ faire$

$prod(i) = 0;$

$pour\ j\ de\ 0$ à $N - 1\ faire$

$prod(i) = prod(i) + mat(i, j) * vect(j);$

$Fin\ pour;$

$Fin\ pour;$

$Retourner * prod\ ;$

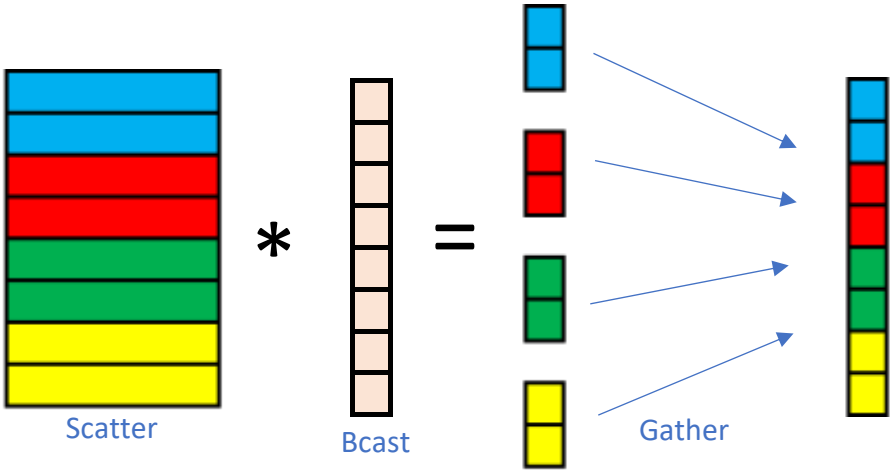
$\underline{Fin;}$

Le produit matrice-vecteur est obtenu en multipliant chaque ligne de la matrice par le vecteur, on multiplie la première ligne de la matrice par le vecteur, le résultat est un réel qui se place dans la première case du vecteur solution, en multipliant la deuxième ligne de la matrice par le vecteur le résultat se place dans la deuxième case du vecteur solution, et ainsi de suite, à la fin on aura comme résultat un vecteur dont la taille est égale au nombre de ligne de la matrice, rappelant que le produit matrice vecteur ne peut se faire que si le nombre de colonne de la matrice ne soit égale au nombre de ligne du vecteur à multiplier.

La version parallèle de se programme est realisé en applicant un MPI_ Scatter sur les lignes de la matrice, chaque processus resevra les élements de la ligne entière, c.à.d que si par exemple on a une matrice de 4 ligne et on a 4 processus sur notre machine, alors le processus 0 recevra la première ligne de la matrice, le processus 1 recevra la deuxième, et ainsi de suite.

Comme dans le cas précédant (produit scalaire), si le réste de la division entre le nombre de ligne de la matrice et le nombre de processus est different de zéro, alors le dernier processus prendra en charge les lignes éxcédante, c.ad que si on a une matrice de 5 ligne et on a 4 processus, alors le processus 0 recevra la premiere ligne, le processus 1 la deuxieme, le processus 2 la troisieme, et enfin le processus 3 recevra la quatrieme et la cinquieme ligne.

Puisque le produit matrice vecteur se fait en multipliant chaque ligne de la matrice par le vecteur, alors tous les processus doivent recevoir le vecteur, cela c'est fait grace à la commande MPI_Bcast. Le resultat est ensuite regrouper dans un seul processus pour affichage grace à la commande MPI_Gather. Il est à signalé qu'on a fait une Barriere avant l'affichage, comme ça l'affichage du vecteur résultat ne peut se faire avant que tous les processus ne terminent leurs calcules.




**Algorithme parallèle :**

*Début:*

*Donnees*: $mat : matrice\ [M][N]\ de\ reel$

$vect : vecteur\ [N]\ de\ reel$

$i, j : entier, variable\ indice\ de\ boucle$

$prod : vecteur\ [M]\ de\ reel, resultat$

$MPI\_Scatter\ (partagage\ ligne\ mat\ aux\ processus)$

$MPI$_Bcast (envoyer vect à tous les processus)

$pour\ i\ de\ 0$ à $M-1\ faire$

$prod(i) = 0;$

$pour\ j\ de\ 0$ à $N-1\ faire$

$prod(i) = prod(i) + mat(i,j) * vect(j);$

$Fin\ pour;$

$Fin\ pour;$

$MPI\_Gather\ (regrouper\ le\ resultat\ dans\ un\ seul\ processus)$

$Stockage$

$MPI\_Barrier$

$Affichage$

*Fin;*

# Produit matrice-matrice :

**Algorithme :**

$\underline{Début}$:

$Donnees$: $mat1 : matrice\ [M][N]\ de\ reel$

$mat2 : matrice\ [N][P]\ de\ reel$

$i, j, k : entier, variable\ indice\ de\ boucle$

$prod : matrice\ [M][P]\ de\ reel, resultat$

$pour\ k\ de\ 0$ à $P - 1\ faire$

$pour\ i\ de\ 0$ à $M - 1\ faire$

$prod(i,k) = 0;$

$pour\ j\ de\ 0$ à $N - 1\ faire$

$prod(i,k) = prod(i,k) + mat1(i,j) * mat2(j,k);$

$Fin\ pour;$

$Fin\ pour;$

$Fin\ pour;$

$Retourner\ ** prod\ ;$

$\underline{Fin;}$

Le produit matrice-matrice est calculé en multipliant chaque ligne de la première matrice par les colonnes de la deuxième matrice, en multipliant chaque ligne de la première matrice par la première colonne de la deuxième matrice, cela nous donnera la première colonne de la matrice résultat, en multipliant chaque ligne de la première matrice par la deuxième colonne de la deuxième matrice, cela nous donnera la deuxième colonne de la matrice résultat et ainsi de suite. Le résultat est une matrice de réels qui a le même nombre de lignes que la première matrice et le même nombre de colonnes que la deuxième. Le produit matrice-matrice ne peut se faire que si le nombre de colonnes de la première matrice ne soit pas égale au nombre de lignes de la deuxième matrice.

Pour la version parallèle, on peut procéder exactement comme le cas du produit matrice vecteur, on distribue les lignes de la première matrice sur les processus, alors on fait un MPI_Scatter sur les lignes de la matrice, chaque processus recevra les éléments de la ligne entière, et pareil si le reste de la division entre le nombre de lignes de la première matrice et le nombre de processus est différent de zéro, alors le dernier processus recevra les lignes restante.

Avant de parler du produit matriciel, il est important de préciser que le stockage de la deuxième matrice a été fait suivant les colonnes. Pour faire le produit, chaque processus doit recevoir les colonnes de la deuxième matrice, alors on fait un MPI_Bcast sur les collone de la deuxième matrice, par exemple si on a 4 processus, la première colonne de la deuxième matrice est reçu par les processus 0, 1, 2 et 3, chaque processus contient une portion différente de la première matrice (lignes de la première matrice), on pourra alors faire le produit et le résultat sera regrouper dans un seul processus

grâce à MPI_Gather, ce résultat représentera la première colonne de la matrice résultat. Pour trouver les autres colonnes de la matrice résultat, il suffit de faire une boucle avant le début du programme de 1 jusqu'au nombre de colonnes de la deuxième matrice pour traiter toutes les colonnes de cette dernière. On fait aussi une Barrière pour éviter d'afficher le résultat avant que tous les processus ne terminent le calcule. Un algorithme est donné pour une meilleure compréhension.

**Algorithme parallèle :**

*Début*:

*Donnees*: $mat1 : matrice\ [M][N]\ de\ reel$

$mat2 : matrice\ [N][P]\ de\ reel$

$i, j, k : entier, variable\ indice\ de\ boucle$

$prod : matrice\ [M][P]\ de\ reel, resultat$

$MPI\_Scatter\ (partagage\ ligne\ mat1\ aux\ processus)$

$pour\ k\ de$ 1 à $P\ faire$

$MPI$_Bcast (envoyer colonne_k de mat2 à tous les processus)

$pour\ i\ de$ 0 à $M - 1\ faire$

$prod(i, k) = 0;$

$pour\ j\ de$ 0 à $N - 1\ faire$

$prod(i, k) = prod(i, k) + mat1(i, j) * mat2(j, k);$

$Fin\ pour;$

$Fin\ pour;$

$MPI\_Gather\ (regrouper\ le\ resultat\ dans\ un\ seul\ processus)$

$Stockage$

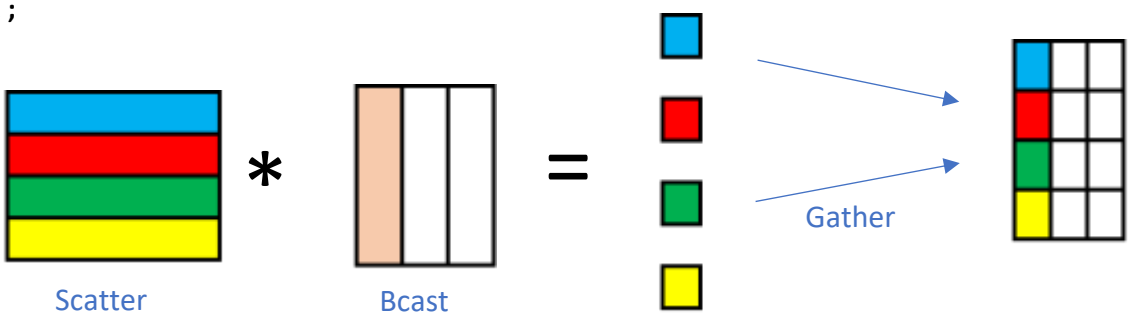
$Fin\ pour;$

$MPI\_Barrier$

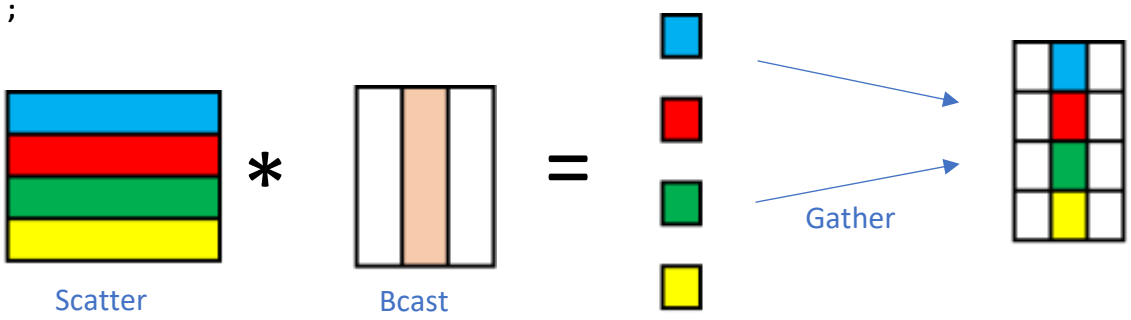
$Affichage$

*Fin*;

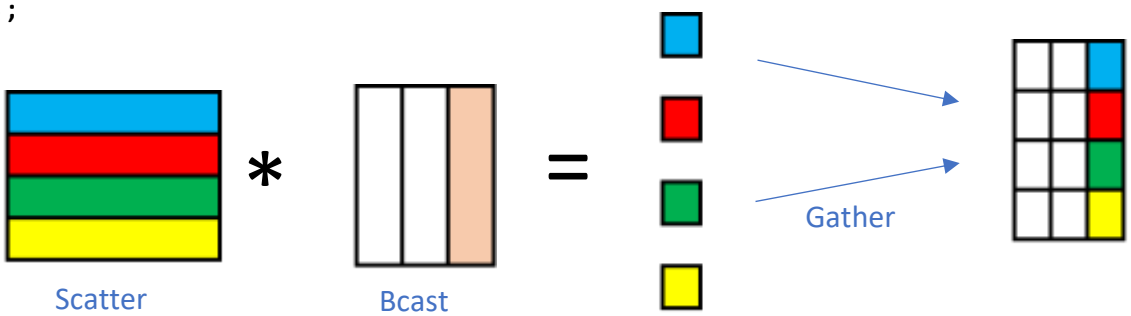
**Pour i = 1 ;**




**Pour i = 2 ;**




**Pour i = 3 ;**




## Analyse de performance

**La Scalabilité** d'un programme parallèle désigne l'augmentation des performances obtenues lorsque l'on ajoute des processus, on va voir dans ce qui suit la scalabilité forte et faible. **La Scalabilité forte** sera déterminer en fixant la taille du problème tout en augmentant le nombre de processus. **La Scalabilité faible** quant à elle sera obtenu en augmentant la taille du problème avec le même nombre de processus.

On constate une bonne scalabilité forte, en fixant le problème et en augmentant le nombre de processus de calcule, lorsqu'on mesure le temps d'exécution avec $time\ mpirun$, on remarque que le temps d'exécution diminue mais très légèrement.

Bien sûr, en fixant le nombre de processus de calcule et en augmentant la taille du problème (Scalabilité faible) le temps d'exécution augmente.