INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES LYON

4BIM

TD1 : OpenMP - Manipulation de matrices et de vecteurs

Alice GENESTIER, Hanâ LBATH

Mars 2019

Table des matières

1 Exercice 1

1.1 Vecteurs

Question 1.9
Remarque : dans cette partie, tous nos temps moyens ont été obtenus en récoltant 4 valeurs de temps par ensemble de conditions.

Dans un premier temps, nous avons effectué un **passage à l'échelle faible**. Pour ce faire, nous avons exécuté notre code avec les conditions suivantes :

— 1 coeur et une taille de vecteur de 10^6
— 2 coeur et une taille de vecteur de $2 * 10^6$
— 4 coeur et une taille de vecteur de $4 * 10^6$

Nous avons obtenu, pour les trois fonctions différentes (addition de deux vecteurs, produit d'un vecteur par un double et somme des éléments d'un vecteur) les temps d'exécutions moyens visibles sur la figure 1 :

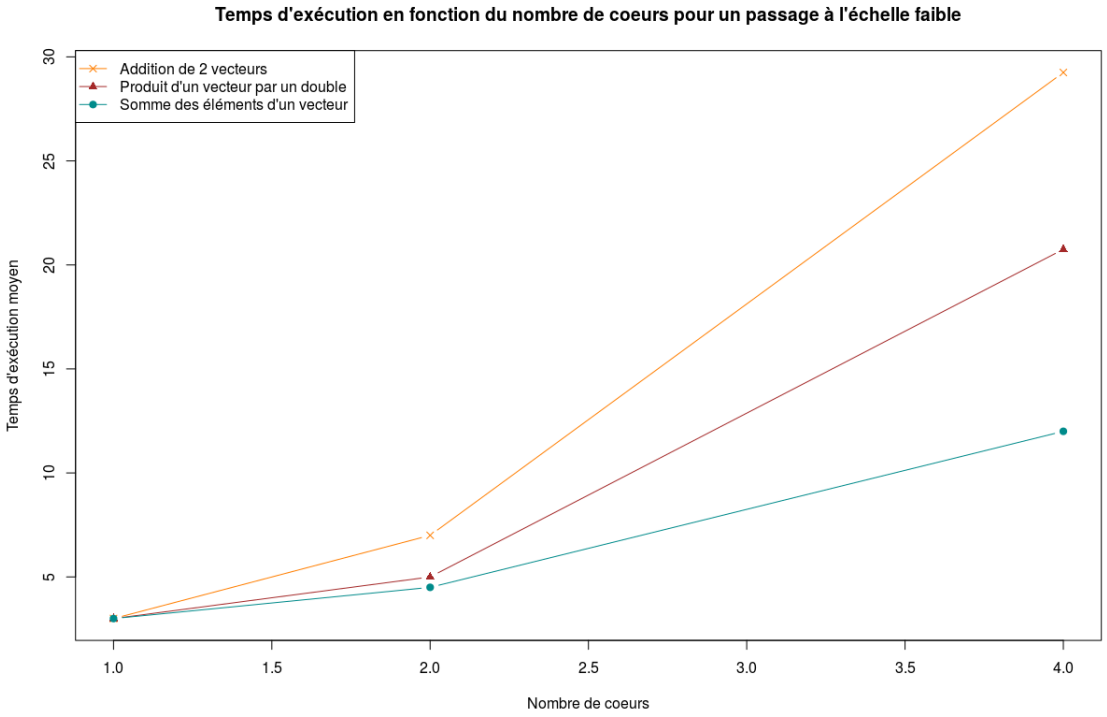


FIGURE 1 – Temps d'exécution en fonction du nombre de coeurs pour trois fonctions différentes

Nous pouvons remarquer qu'additioner deux vecteur est la fonction la plus lente en terme de temps d'exécution, ce qui semble logique puisqu'on doit parcourir deux vecteurs, tandis que pour les autres fonctions on ne parcourt qu'un seul vecteur.

Nous pouvons en outre noter que, contrairement à ce à quoi on s'attendait, plus on ajoute de coeurs par passage à l'échelle faible, plus le temps d'exécution augmente. Cela est probablement dû au fait que les calculs effectués ne sont pas suffisemment complexes et/ou les données ne sont pas suffisemment grandes pour que le temps gagné en parallélisant les calculs l'emporte sur le temps perdu à communiquer.

Nous avons ensuite réalisé un **passage à l'échelle forte**. Pour ce faire, nous avons exécuté notre code avec 1, 2 ou 4 coeurs et, pour chaque nombre de coeurs, une taille de vecteur de 10^6, 10^7 ou 10^8.

Les temps d'exécution moyens obtenus pour *l'addition de deux vecteurs* sont représentés dans la figure 2.

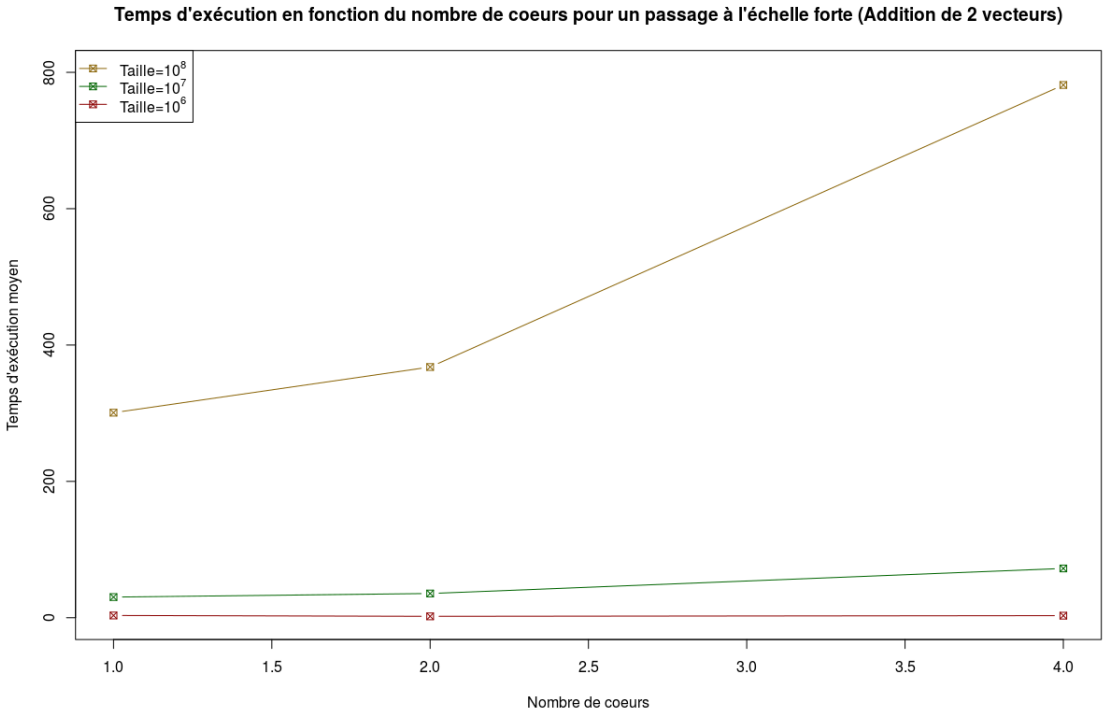


FIGURE 2 – Temps d'exécution en fonction du nombre de coeurs pour l'addition de deux vecteurs, pour des tailles de vecteurs différentes

De la même façon, nous avons obtenu les temps d'exécutions moyens pour *le produit d'un vecteur par un scalaire* :

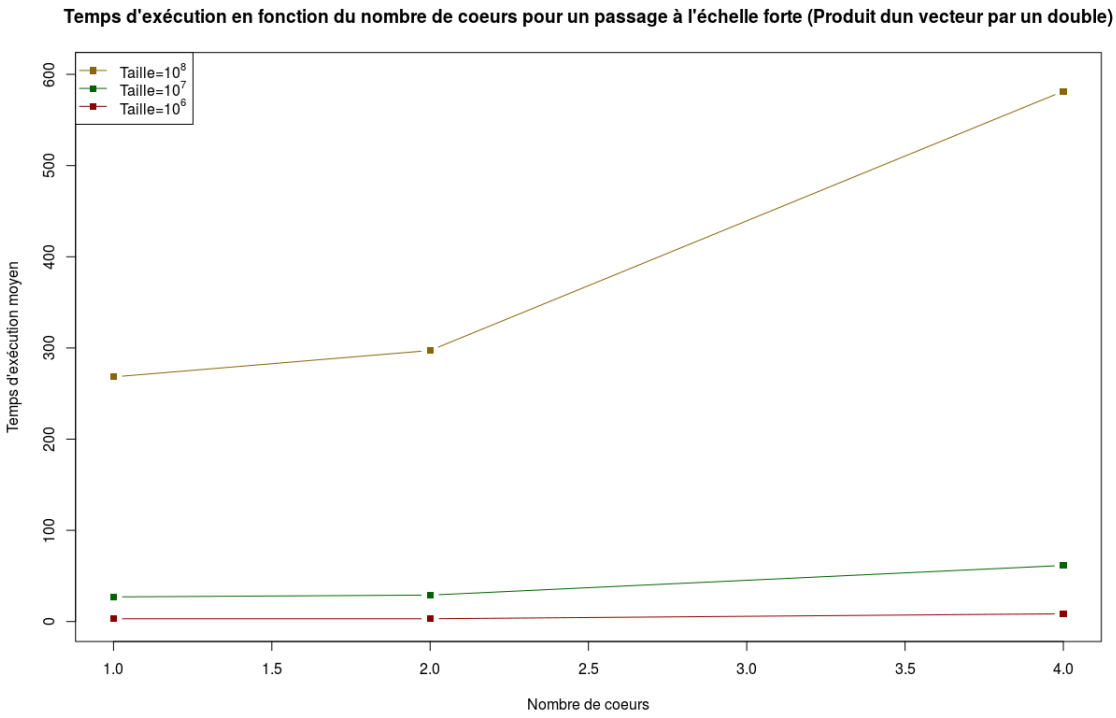


FIGURE 3 – Temps d'exécution en fonction du nombre de coeurs pour produit d'un vecteur par un double, pour des tailles de vecteurs différentes

Les temps d'exécutions moyens obtenus pour *la somme des éléments d'un vecteur* sont représentés dans la figure 4 :

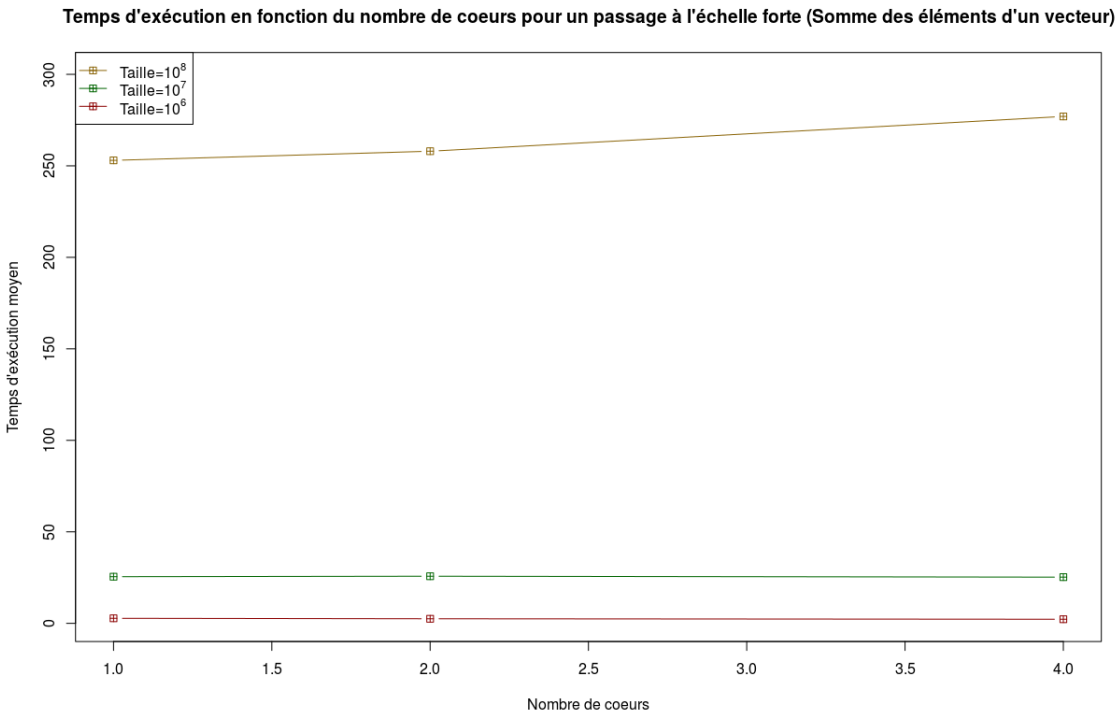


FIGURE 4 – Temps d'exécution en fonction du nombre de coeurs pour la somme des éléments d'un vecteur, pour des tailles de vecteurs différentes

De la même façon que pour le passage à l'échelle faible, nous pouvons remarquer que, pour toutes les fonctions et pour toutes les tailles de vecteur, le temps d'exécution, soit reste du même ordre de grandeur, soit augmente, lorsque l'on augmente le nombre de coeur.

Nous pouvons aussi comparer, à taille de vecteur constante, les temps d'exécution moyens pour ces trois fonctions :

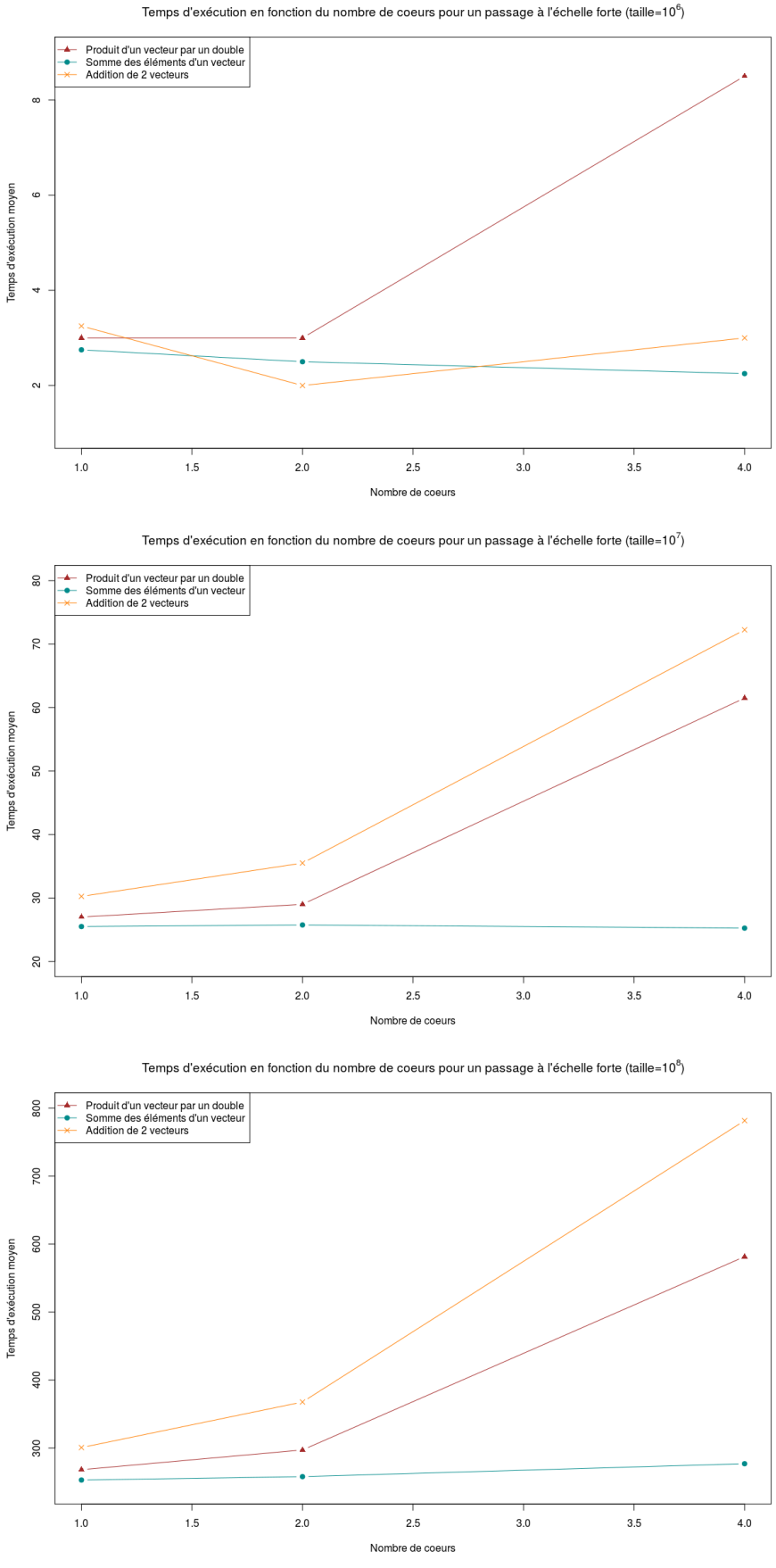


FIGURE 5 – Temps d'exécution en fonction du nombre de coeurs pour différentes tailles de vecteurs (10^6, 10^7, 10^8)

1.2 Matrices

Question 1.10
Remarque : dans cette partie, tous nos temps moyens ont été obtenus en récoltant 4 valeurs de temps par ensemble de conditions.

Dans un premier temps, nous avons effectué un **passage à l'échelle faible**. Pour ce faire, nous avons exécuté notre code avec les conditions suivantes :
- 1 coeur et une taille de matrice de 1000
- 2 coeur et une taille de matrice de 2000
- 4 coeur et une taille de matrice de 4000

Nous avons obtenu, pour les trois fonctions différentes (addition de deux matrices, produit d'une matrice par un double et somme des éléments d'une matrice) les temps d'exécutions moyens visibles sur la figure 6 :

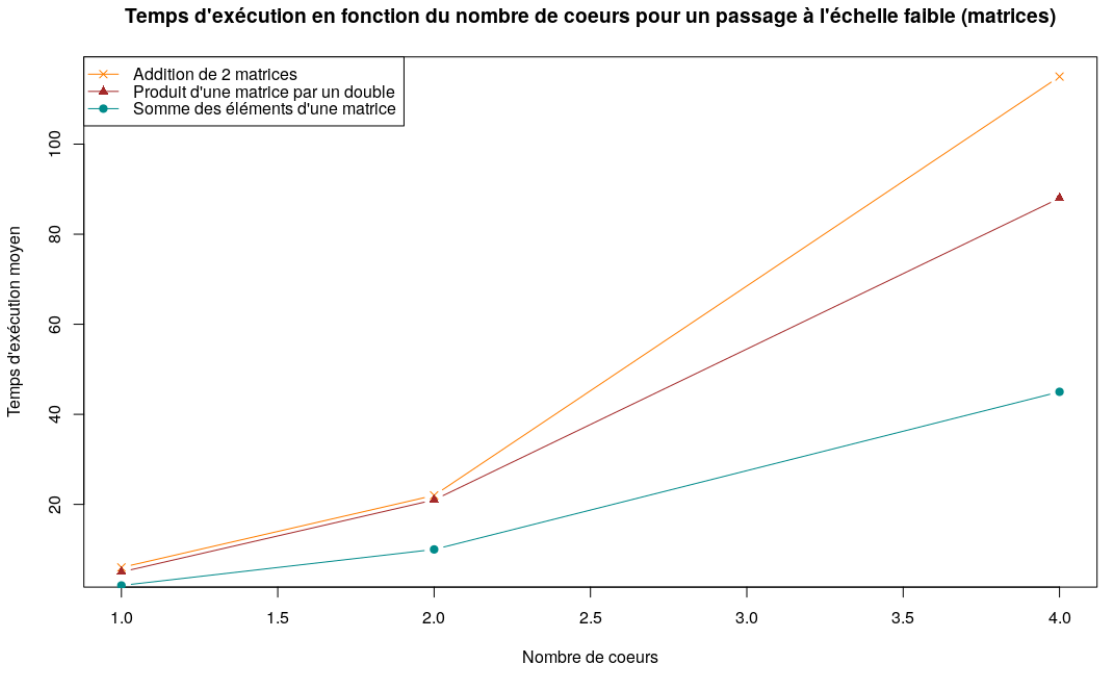


FIGURE 6 – Temps d'exécution en fonction du nombre de coeurs pour trois fonctions différentes

Nous pouvons remarquer que, de la même façon que pour les vecteurs, additioner deux matrices est la fonction la plus lente en terme de temps d'exécution.
De même que pour les vecteurs, plus on ajoute de coeurs par passage à l'échelle faible, plus le temps d'exécution augmente.

Nous avons ensuite réalisé un **passage à l'échelle forte**. Pour ce faire, nous avons exécuté notre code avec 1, 2 ou 4 coeurs et, pour chaque nombre de coeurs, une

taille de vecteur de 50, 500 ou 5000.

Les temps d'exécution moyens obtenus pour *l'addition de deux matrices* sont représentés dans la figure 7.

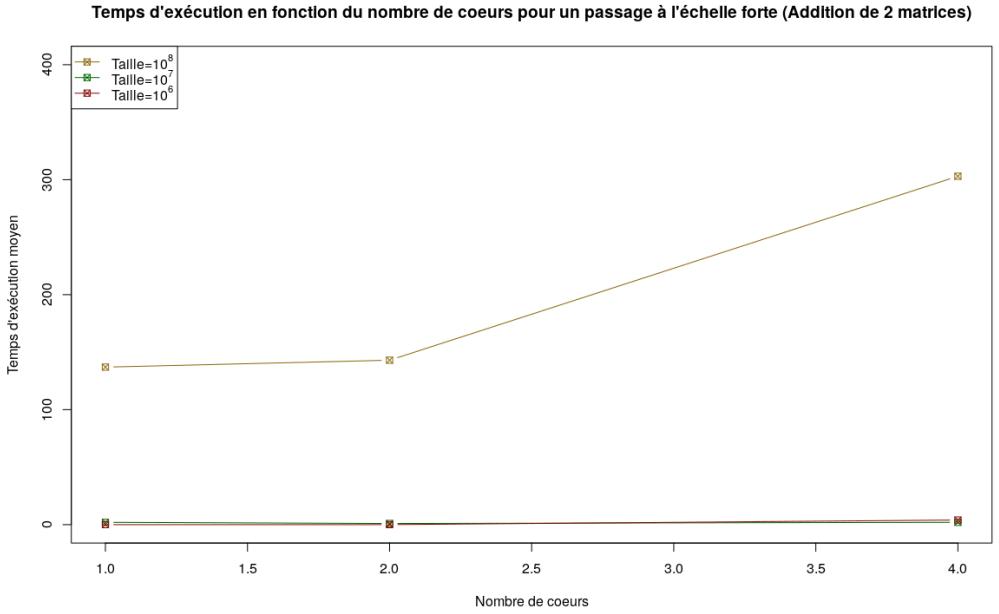


FIGURE 7 – Temps d'exécution en fonction du nombre de coeurs pour l'addition de deux matrices, pour des tailles de matrices différentes

De la même façon, nous avons obtenu les temps d'exécutions moyens pour *le produit d'une matrice par un scalaire* :

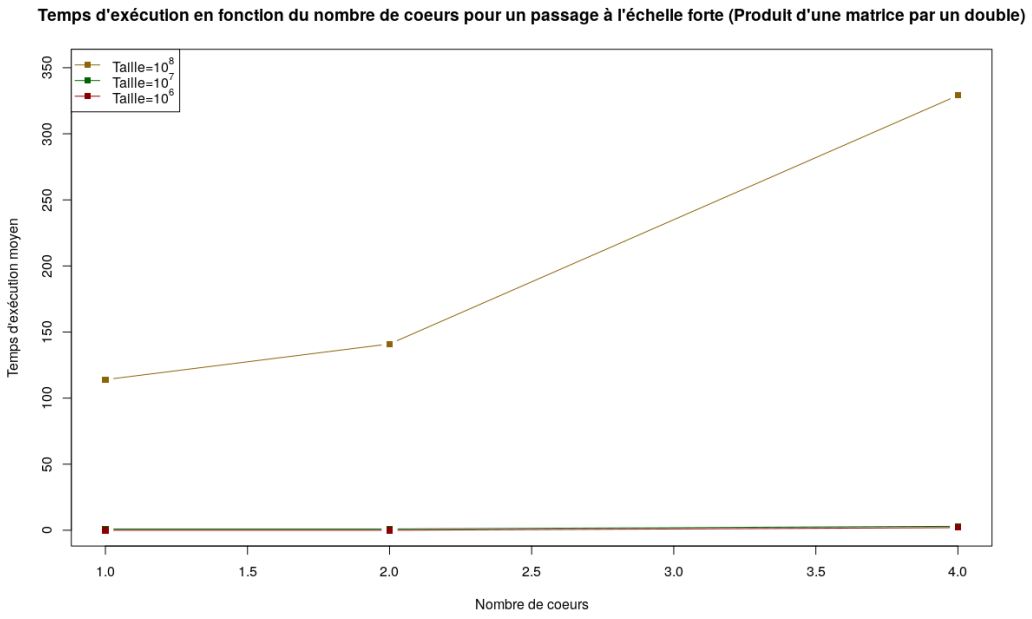


FIGURE 8 – Temps d'exécution en fonction du nombre de coeurs pour le produit d'une matrice et d'un scalaire, pour des tailles de matrices différentes

Les temps d'exécutions moyens obtenus pour *la somme des éléments d'une matrice* sont représentés dans la figure 9 :

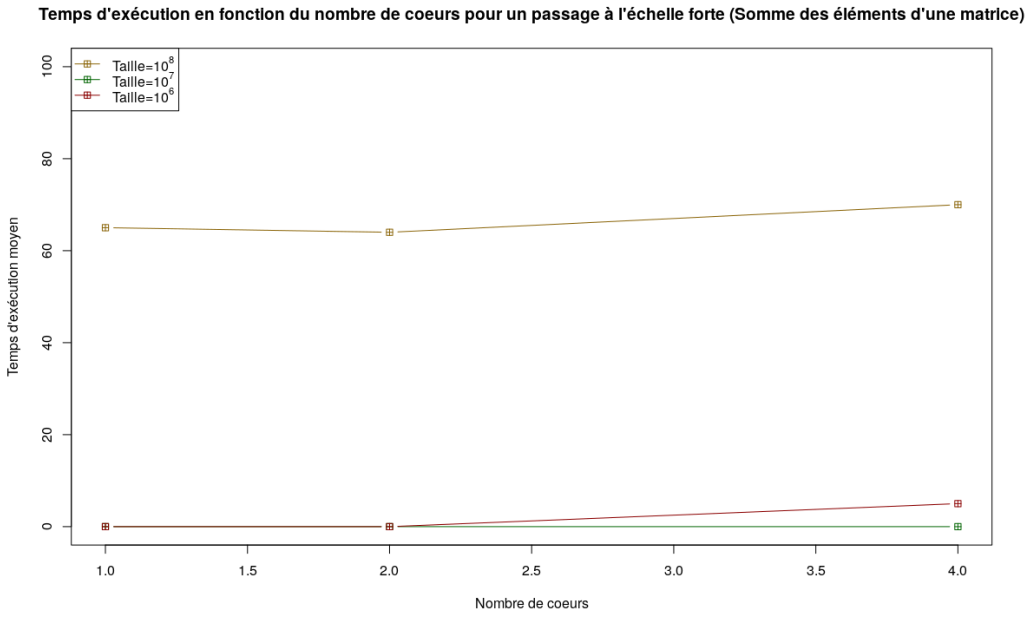
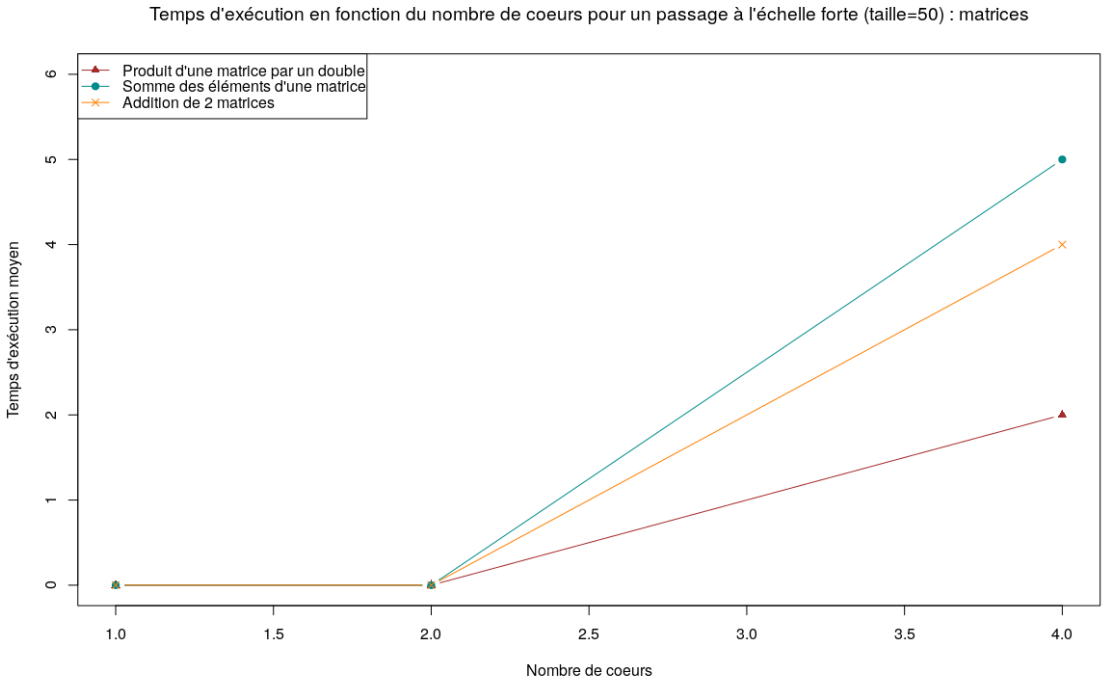
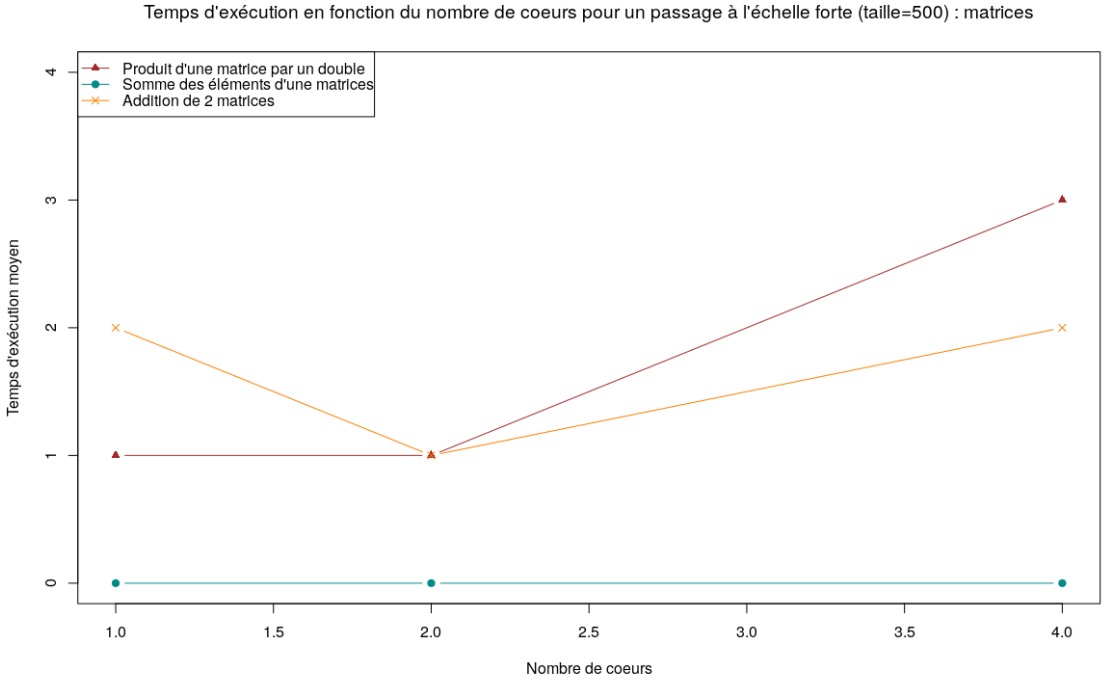


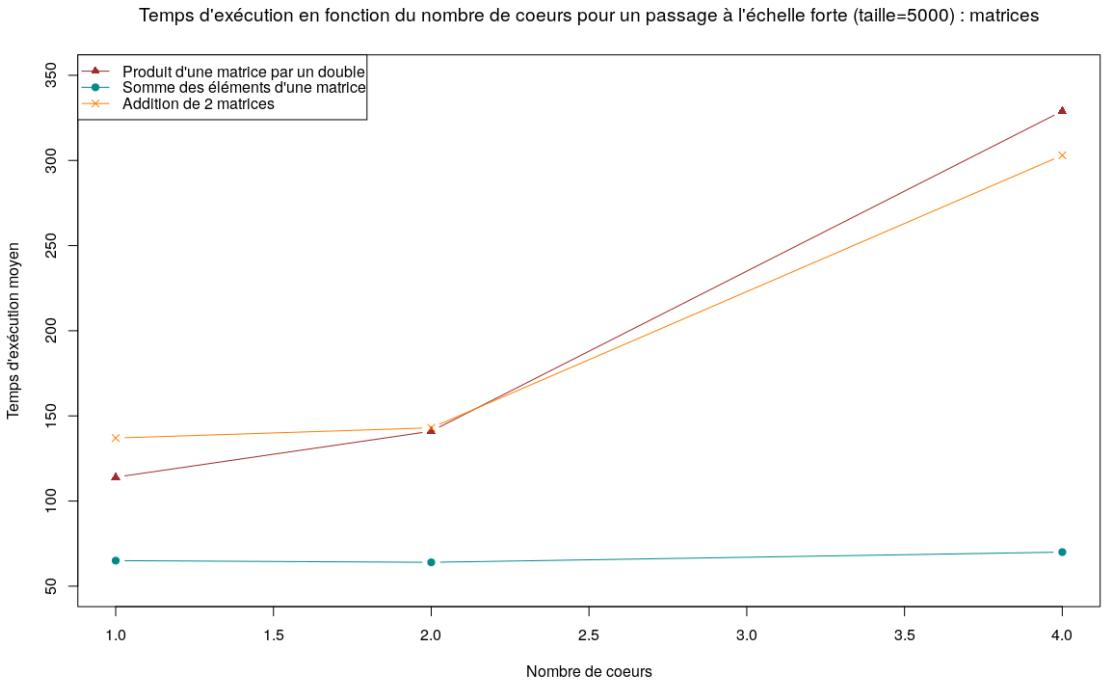
FIGURE 9 – Temps d'exécution en fonction du nombre de coeurs pour la somme des éléments d'une matrice, pour des tailles de matrices différentes

Nous pouvons aussi comparer, à taille de matrice constante, les temps d'exécution

moyens pour ces trois fonctions :







De la même façon que pour le passage à l'échelle faible, nous pouvons remarquer que, pour toutes les fonctions et pour toutes les tailles de matrices, le temps d'exécution, soit reste du même ordre de grandeur, soit augmente, lorsque l'on augmente le nombre de coeur (sauf dans le cas de l'addition de deux matrices de taille 500 où on observe une légère amélioration pa rapport aux autres fonctions).

2 Compter des lettres

Nous avons dans un premier temps écrit un code qui permettait de compter le nombre d'occurences de chaque lettre présente dans une matrice. Cette fonction créait un objet map qui répertoriait uniquement les lettres présentes dans la matrice. Cependant, quand nous avons voulu paralléliser ce programme, nous nous sommes apperçues qu'il était plus simple d'initialiser l'objet map en le remplissant avec l'ensemble des lettres de l'alphabet. En outre, pour des matrices suffisamment grandes, statistiquement, toutes les lettres de l'alphabet sont présentes dans ces dernières.