Compte rendu: Projet , produit de deux matrices avec la méthode de Strassen.

Tae Young OH et RAKOTOARIJAONA Andrianarivo.

Problématique Le but de ce projet est de programmer l'algorithme de Strassen en C++ et de le comparer par rapport à l'algorithme de produit matricielle naïf.« Quand est ce que l'algorithme de Strassen est plus efficace que le produit naïf ? »

1 Classe utilisé :

Pour commencer, On a crée dans un fichier matrice.hpp une classe matrice qui contient : En privé un entier qui va être la taille de la matrice et un pointeur pour faire un tableau. Après dans le champs publique il y a en premier lieu :

— Les accesseurs qui sont

1. `double& operator()(int i, int j)` : Qui permet d'écrire dans la ligne i colonne j.
2. `double operator()(int i, int j)const` :Qui permet de lire dans la ligne i colonne j.

— un constructeur par défaut `matriceint N` qui ne prend en paramètre que la taille de la matrice initialisé par défaut à $N = 4$(matrice de taille 4x4) et qui définit une matrice identité.

— Vu qu'on manipule des pointeurs, on va dans ce faire le constructeur par copie, l'opérateur d'affectation, et le destructeur .

— Les opérateurs pour faire le produit, la somme et la soustraction de deux matrices qui sont des `friend` de la classe matrice, et un opérateur $-A$ où A est une matrice.

— Les opérateurs $*=$, $+=$ et $-=$.

— L'opérateur $<<$ pour afficher une matrice, et une méthode `int get_n(void)` qui permet de lire la taille d'une matrice si besoin est.

1.1 L'algorithme de Strassen :

Dans un fichier strassen.cpp, on a fait l'algorithme de strassen en utilisant la classe matrice, et en faisant en sorte que l'algorithme de strassen commence à partir d'une matrice de taille $n > 16$.Pour l'algorithme (voire le fichier).

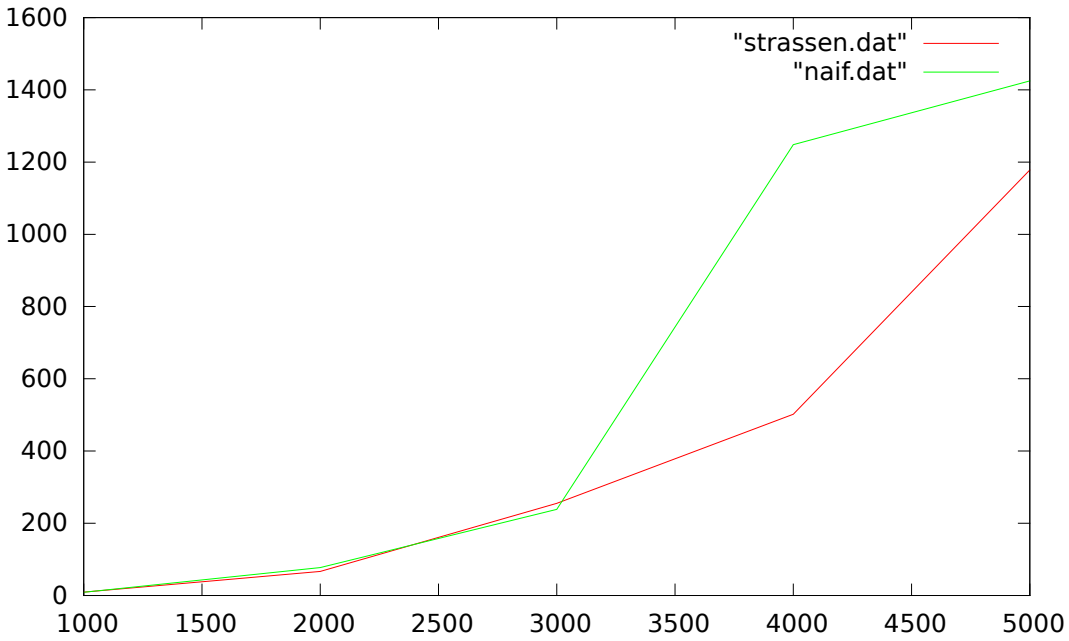
1.1.1 Remarque :

On a fait un fichier main.cpp pour vérifier si les résultats du produit matriciel naïf et Strassen donnent les mêmes résultats.

1.2 Comparaison entre Strassen et le produit matriciel naïf :

Tout d'abord, on a crée deux fichier test1.cpp et test2.cpp dont la première fichier va contenir l'utilisation de l'algorithme de Strassen et l'évaluation du temps de son exécution pour le produit de matrice de différente taille.Après on récupère le temps d'exécution pour chaque taille donné. Dans le deuxieme fichier, on évalue le produit naïf et faisant la même chose que dans le fichier test1.cpp.

On a fait la comparaison avec des matrices de grande taille, par exemple pour des matrices de taille 1000 à 5000 de pas égale à 1000.



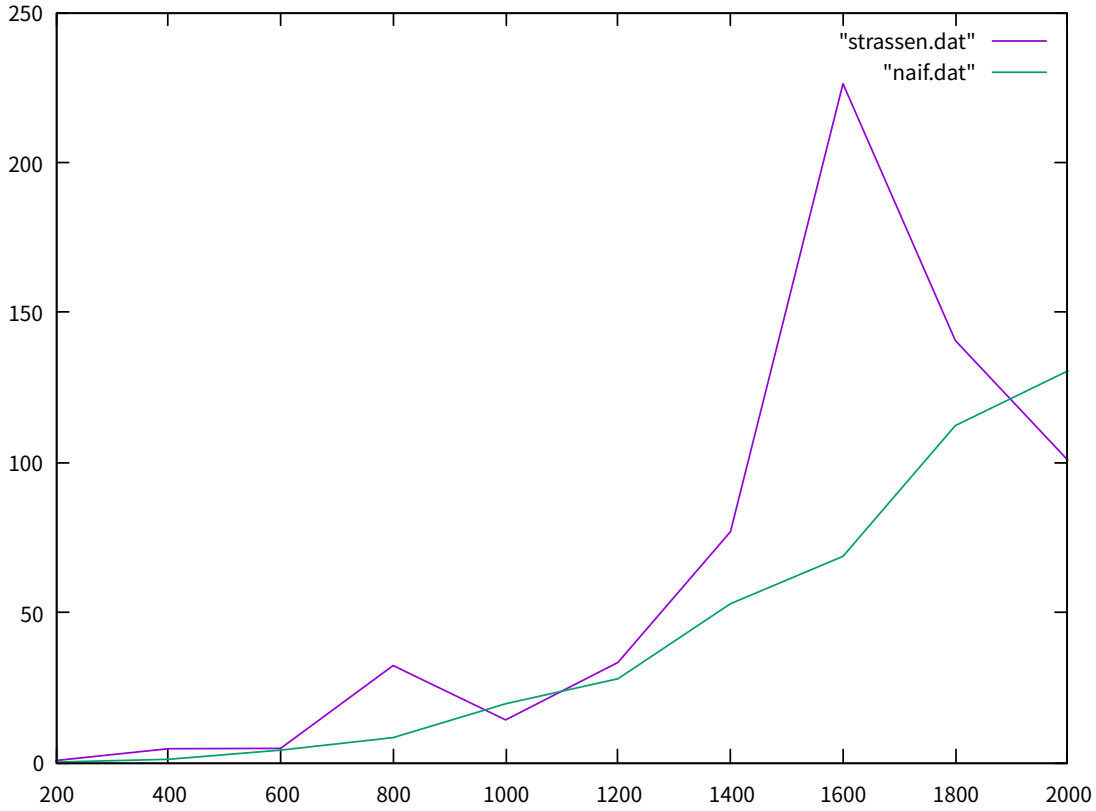
Dans ce graphe, sur l'axe des abscisses, on a les différentes valeur de n et sur l'axe des ordonnée, le temps d'exécution cpu en seconde.

1.2.1 Constatation :

On peut dire selon la courbe de pour $n > 3000$ l'algorithme de Strassen est plus rapide que le produit naïf pour les différentes valeurs de n qu'on a pris.

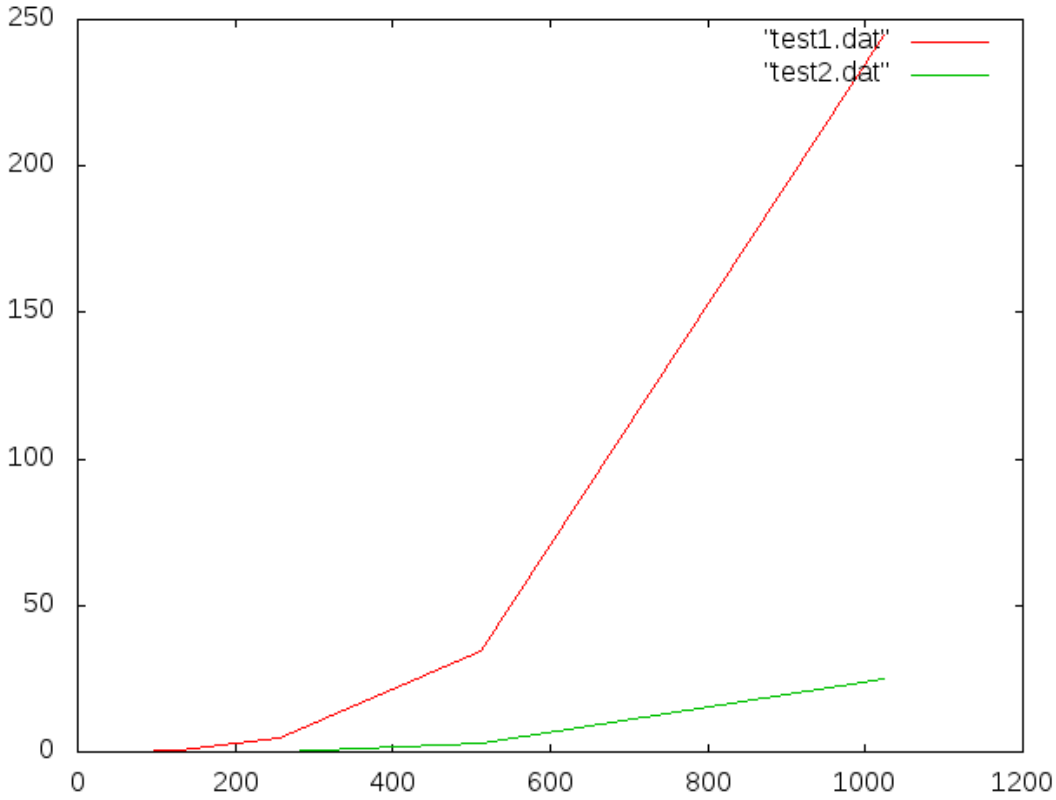
1.3 Inconvénient de l'algorithme de Strassen :

On a essayer d'appliquer Strassen avec des cas où n est entre 200 et 2000 avec un pat de 400 et cela nous donne :



On remarque que des fois pour un n multiple de 2 Strassen est bon, mais pour certain n cela prend trop de temps.

Du fait que cette algorithme est récursif, on pense que cela affecte à sa vitesse d'exécution, d'où, pour certain valeur de n, qui sont de la forme 2^m, Strassen est plus long, car l'algorithme est obligé de ré-appliquer Strassen dans sur les matrices de tailles $\frac{n}{2}$ plusieurs fois dû au fait de la récursivité.



Dans ce graphe, sur l'axe des abscisses, on a les différentes valeur de n et sur l'axe des ordonnée, le temps d'exécution cpu en seconde.Et test1 correspond à Strassen, test2 correspond au produit naïf. Et cela montre que Strassen est lent quele produit naïf.

Constatation :

Lorsqu'on peut diviser n par 2^m avec $m \in \{1, 2, 3, 4\}$, et qu'il existe q avec $q \equiv 1 mod[2]$ tel que $n = q2^m$, Strassen est beaucoup plus rapide.Par contre, pour $m > 4$, on remarque sur la deuxième courbe que Strassen commence à être plus long que le produit naïf.