**INF8775 – Analyse et conception d’algorithmes**

TP1 – Hiver 2023

| **Nom, prénom, matricule des membres** | MESSEDI, Mohamed-Ali, 1946597  CARON, Thomas, 1944066 |
| --- | --- |
| **Note finale / 13** | 0 |

**Informations techniques**

* Répondez directement dans ce document .docx. Veuillez ne pas inclure le texte en italique servant de directive. La correction se fait sur ce même rapport.
* Vous devez faire une remise électronique sur Moodle avant le 14 février à 23h55 pour le groupe B2 et 21 février à 23h55 pour le groupe B1 en suivant les instructions suivantes :
  + Vos fichiers doivent être remis dans une archive zip à la racine de laquelle on retrouve :
    - Ce rapport sous format docx.
    - Un script nommé *tp.sh* servant à exécuter les différents algorithmes du TP. L’interface du script est décrite à la fin du rapport.
    - Le code source et les exécutables.
* Vous avez le choix du langage de programmation utilisé mais vous devrez utiliser les mêmes langage, compilateur et ordinateur pour toutes vos implantations. Notez que le code et les exécutables soumis seront testés sur les ordinateurs de la salle L-4714 et doivent être compatibles avec cet environnement. En d’autres mots, tout doit fonctionner correctement lorsque le correcteur exécute votre script *tp.sh* sur un des ordinateurs de la salle.
* Si vous utilisez des extraits de codes (programmes) trouvés sur Internet, vous devez en mentionner la source, sinon vous serez sanctionnés pour plagiat.
* On vous encourage à lire le guide intitulé « guide bash » sur Moodle pour faire vos graphiques. C’est un guide qui a été conçu pour un ancien TP, mais il contient beaucoup d’informations utiles.

**Mise en situation**

Ce travail pratique se répartit sur deux séances de laboratoire et porte sur l’analyse empirique et hybride des algorithmes. Dans les capsules vidéo de la semaine 3, trois approches d’analyse de l’implantation d’un algorithme sont décrites. Vous les mettrez en pratique pour des algorithmes de résolution d’un problème connu.

**Implantation**

Vous implanterez les algorithmes de multiplication de matrices *conventionnel* et *diviser-pour-régner* (algorithme de *Strassen*). Vous ferez deux versions de ce dernier, avec et sans un seuil de récursivité déterminé expérimentalement par essai-erreur. Pour la version avec seuil de récursivité, les (sous-)exemplaires dont la taille est en deçà de ce seuil ne seront plus résolus récursivement mais plutôt directement avec l’algorithme conventionnel. Assurez-vous que vos implantations sont correctes en comparant les résultats des trois algorithmes.

**Jeu de données**

Vous travaillerez avec des matrices de taille 2*N* × 2*N*. Pour chaque valeur de *N*, vous devrez générer cinq matrices que vous pourrez multiplier deux à deux, ce qui vous donnera dix exemplaires. Utilisez au moins cinq valeurs consécutives de *N* pour votre analyse, ce choix pourra varier d’une équipe à l’autre selon la qualité de vos implémentations.

Vous trouverez dans l’archive du TP un script python *inst\_gen.py* servant à générer les exemplaires. Ce script s’exécute de la manière suivante :

inst\_gen.py -S TAILLE\_MIN [-t NB\_TAILLES] [-n NB\_EXEMPLAIRES] [-r RANDOM\_SEED]

TAILLE\_MIN correspond à la plus petite valeur de N que vous voudrez utiliser

NB\_TAILLES correspond au nombre de tailles consécutives que vous voulez générer (par exemple si TAILLE\_MIN = 2 et NB\_TAILLES = 3, alors le script génèrera des matrices pour N = 2, N = 3 et N = 4.

NB\_EXEMPLAIRES correspond au nombre de matrices que vous voulez générer pour chaque taille

RANDOM\_SEED correspond à la *seed* utilisée pour la génération aléatoire des matrices

Les fichiers générés débutent avec la valeur de N sur la première ligne et les lignes suivantes correspondent aux lignes de la matrice où chaque nombre est séparé par une tabulation. Voici un exemple pour *N* = 2 :

2

1 3 2 1

0 1 2 2

3 3 3 1

3 0 1 1

**Présentation des résultats**

|  | / 4 pts |
| --- | --- |

**Tableau des résultats**

| **Algorithme** | **Taille de la matrice** | **Taille du seuil** | **Temps d'exécution (secondes)** |
| --- | --- | --- | --- |
| Classique | 16 | - | 0.000664405 |
| Classique | 32 | - | 0.00395641 |
| Classique | 64 | - | 0.0298926 |
| Classique | 128 | - | 0.283076 |
| Classique | 256 | - | 2.07886 |
| Classique | 512 | - | 16.669 |
| Classique | 1024 | - | 141.375 |
| Strassen | 16 | - | 0.00563711 |
| Strassen | 32 | - | 0.030539 |
| Strassen | 64 | - | 0.223348 |
| Strassen | 128 | - | 1.56778 |
| Strassen | 256 | - | 11.7314 |
| Strassen | 512 | - | 81.4587 |
| Strassen | 1024 | - | 609.188 |
| Strassen avec Seuil | 16 | 64 | 0.000575237 |
| Strassen avec Seuil | 32 | 64 | 0.00354859 |
| Strassen avec Seuil | 64 | 64 | 0.0330758 |
| Strassen avec Seuil | 128 | 64 | 0.233496 |
| Strassen avec Seuil | 256 | 64 | 1.80185 |
| Strassen avec Seuil | 512 | 64 | 12.7839 |
| Strassen avec Seuil | 1024 | 64 | 106.172 |

Tous les temps d’exécutions ont été pris avec l’utilisation de la librairie Python ***timeit*** qui nous permet de calculer le temps d'exécution d’une fonction avec l’aide d’un “performance counter”.

**Tests de puissance***.*

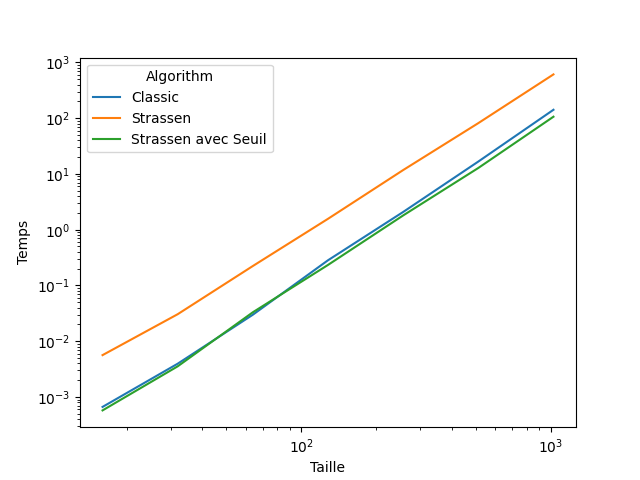
**

Figure 1 : Test de puissance pour chaque algorithme

**Test du rapport**

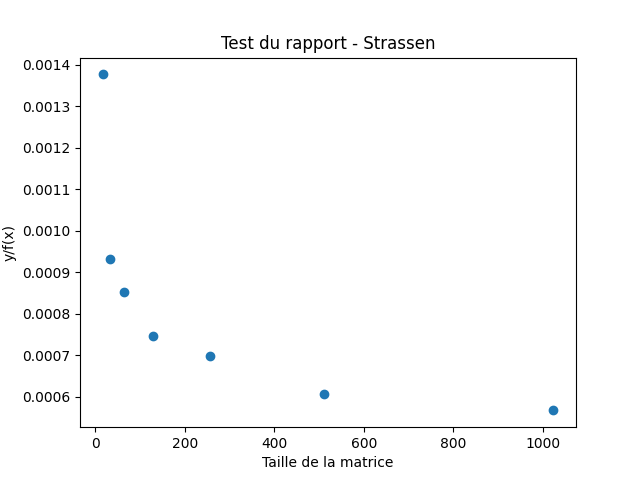
**

Figure 2 : Test du rapport - Méthode Strassen

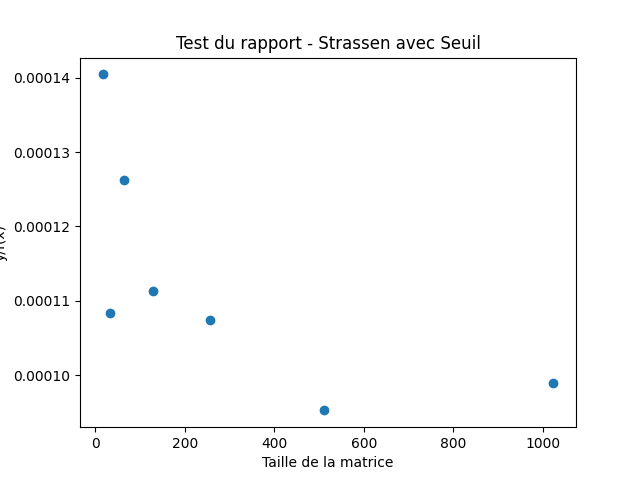
**

Figure 3 : Test du rapport - Méthode Strassen avec Seuil

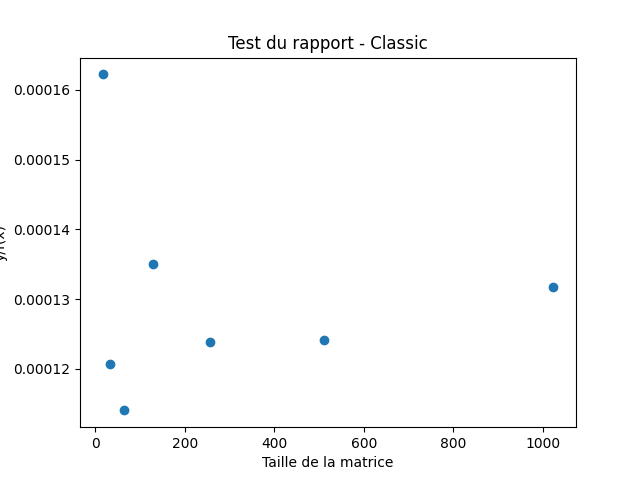
**

Figure 4 : Test du rapport - Méthode Classique

**Test des constantes**

Pour chacun des graphiques ci-dessous, le temps est exprimé en secondes.

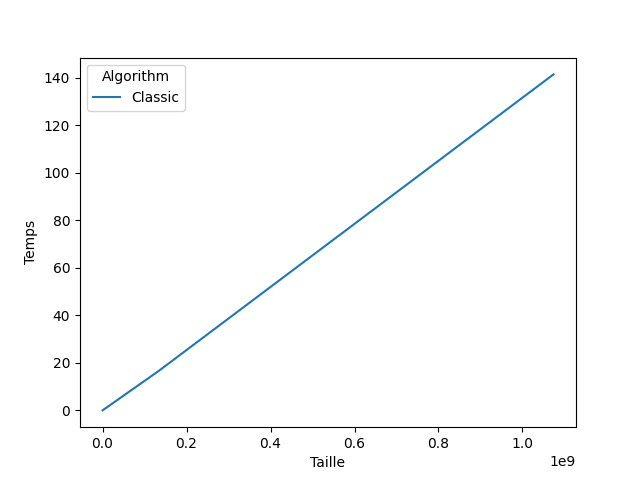


Figure 5 : Test des constantes - Méthode classique

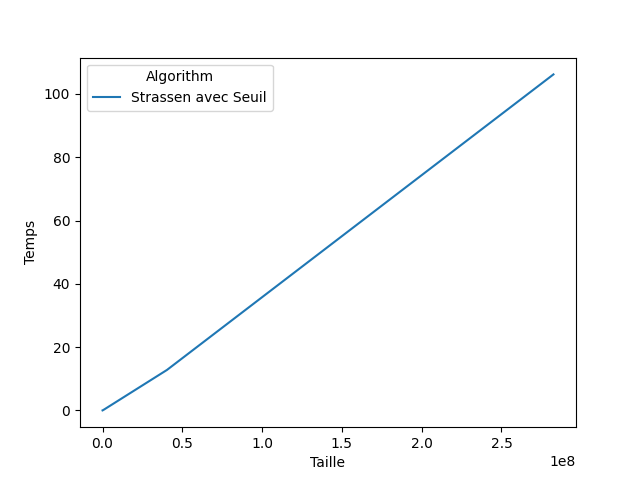


Figure 6 : Test des constantes - Méthode Strassen avec seuil

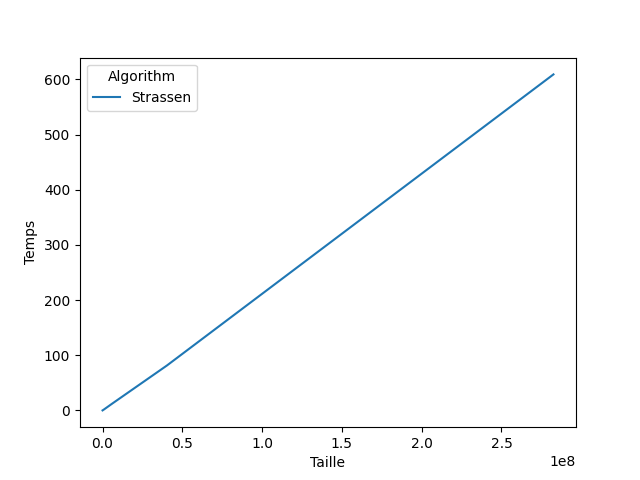


Figure 7 : Test des constantes - Méthode Strassen

**Analyse et discussion**

|  | / 6 pt |
| --- | --- |

**Que pouvez-vous déduire du test de puissance ?**

Le test de puissance est une bonne façon pour nous de déterminer un facteur lié à la croissance polynomiale de notre fonction.

Il est possible de reprendre l'équation vue en cours qui permet de déterminer les facteurs de la croissance de notre algorithme. En l'exprimant sous sa forme logarithmique, on obtient :

Il s'agit d'une forme de fonction avec m étant la pente et b l'ordonnée à l'origine. La valeur m est donc le facteur polynomial de notre fonction.

**Citez la consommation théorique du temps de calcul pour les algorithmes, en notation asymptotique.**

Pour l'algorithme de calcul matriciel conventionnel, on a une complexité de Θ(n³).

Pour obtenir l'algorithme de Strassen, on peut facilement utiliser le Master Theorem pour trouver la constante.

On peut l'exprimer comme suit : T(n) = 7T(n/2) + c(n^2)

De cette expression on peut déduire que l'algorithme de Strassen a une complexité de Θ(nlog2(7)) = Θ(n2.8074)

L'algorithme de Strassen, donc, donne une complexité de Θ(n2.8074)

**Que pouvez-vous déduire du test du rapport ?**

Le test du rapport nous permet, en partant de l'hypothèse que notre fonction s'apparente à une fonction, de trouver si on obtient une convergence vers une constante. S'il n y a pas de convergence, on peut déterminer si on a fait une sous-estimation ou une sur-estimation.

S'il y a bel et bien une convergence, c'est cette valeur qui est la constante multiplicative liée à l'ordre de grandeur de la fonction pour l'hypothèse initiale.

Avec le test de puissance, en comparant l'évolution des courbes on peut voir que Strassen est légèrement moins performant que ses deux autres concurrents, Strassen avec seuil et classique.

Pour la méthode Strassen, on a une convergence vers une valeur de constante de 0.0006.

Pour la méthode Strassen avec seuil, on a une valeur de convergence d'environ 0.0001, et avec la méthode classique on a une valeur d'environ 0.00013.

Ces valeurs nous montrent que la constante multiplicative est sensiblement plus élevée pour Strassen, avec des valeurs plus basses et similaires entre les approches Strassen avec seuil et Classique.

On voit donc que le test du rapport nous donne une évolution plus ou moins similaire en termes de résultats, et c'est ce que les graphiques nous confirment.

**Que pouvez-vous déduire du test des constantes ?**

Avec le test des constantes, on peut déterminer si les résultats trouvés expérimentalement donnent une croissance similaire à celle émise dans l'hypothèse initiale, donc en théorie.

Il est possible de voir que pour toutes les méthodes, on a une droite linéaire, ce qui veut dire qu'on a un coefficient de croissance qui ne varie pas pour chaque algorithme (car la pente de chaque droite reste la même). C'est aussi un indicateur d'une forte corrélation entre les résultats expérimentaux et l'hypothèse théorique.

Le test des constantes nous a permis au départ de trouver que nos résultats n'étaient pas très précis car comme peu de données étaient utilisées, nous avions obtenu une courbe et non une droite, ce qui nous a poussé à raffiner et à augmenter notre jeu de données pour aboutir à un meilleur résultat.

**Discutez de l’impact du seuil de récursivité.**

Ici, on fait un test de la performance de l'algorithme de Strassen en fonction de 7 différents seuils de récursivité. On peut voir que la valeur de seuil et le temps d'exécution ne sont pas dans le même ordre. En effet, on obtient la comparaison suivante :

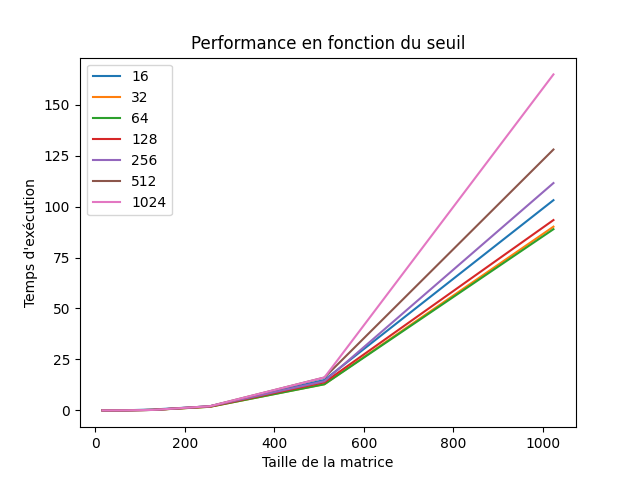
****

Figure 8 : Performance Strassen avec seuil pour différents seuils

**T(64)** < T(32) < T(128) < T(16) < T(256) < T(512) < T(1024)

On peut voir que les meilleurs temps d'exécution sont pour des seuils de 64 et 32 à très peu près, avec la valeur de 64 étant la meilleure.

En effet, il s'agit de trouver le meilleur compromis entre la portion de la matrice ou la méthode Strassen sera utilisée et celle ou en dessous du seuil on utilisera l'algorithme classique.

Les deux méthodes doivent fonctionner ensemble. Trop solliciter l'une ou l'autre résulte d'une mauvaise performance, et il semble que le bon équilibre est pour un seuil autour de 64.

Même si pour une taille de 32 on obtient presque le même temps d'exécution, on gardera la valeur de 64 comme résultat final car c'est sensiblement la plus basse.

**Suite à cette analyse, indiquez sous quelles conditions (taille d’exemplaire ou autre) vous utiliseriez chacun de ces algorithmes. Justifiez.**

Tout d'abord, nous décidons d'éliminer Strassen sans seuil, car sa performance est nettement inférieure à l'approche ou un seuil est introduit.

Après nos recherches, il semblerait que pour un seuil de 64, l'algorithme de Strassen a la meilleure performance. Ainsi, si on garde ce seuil, l'algorithme appellera directement la partie de son programme responsable du calcul par méthode classique à ce niveau là.

Quand on compare respectivement les temps d'exécution entre la méthode Strassen avec seuil de 64 et la méthode classique sur des matrices de tailles 32 et 64, on a des valeurs très similaires.

Ainsi, pour les matrices de taille inférieure à 64, il vaut mieux appeler la méthode classique pour appliquer directement l'algorithme de base, étant donné que faire appel à Strassen reviendrait tout simplement à avoir le même comportement.

Cependant, dès que la taille de matrice est supérieure à ce seuil (128, 256, 512..) on constate que Strassen avec seuil reprend le dessus en matière de performance. C'est donc ce dernier qu'il faut favoriser.

Il est à noter que la décision est influencée par les valeurs expérimentales, qui pourraient être influencées par la machine utilisée, la qualité des algorithmes, l'implémentation, et de nombreux autres facteurs pouvant introduire des incertitudes.

**Autres critères de correction**

**Respect de l’interface** tp.sh

|  | / 1 pt |
| --- | --- |

Utilisation :

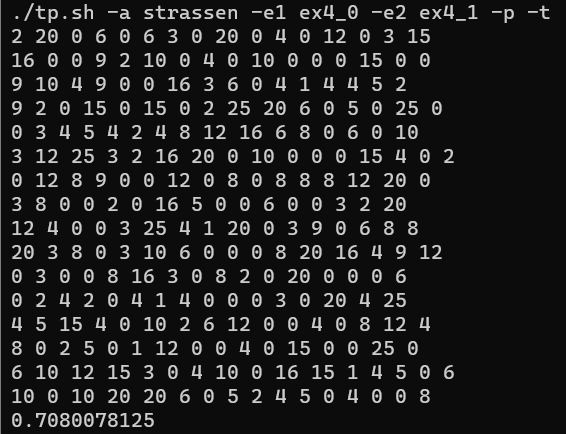
$ ./tp.sh -a {conv, strassen, strassenSeuil} -e1 PATH\_VERS\_EX\_1 -e2 PATH\_VERS\_EX\_2 [-p] [-t]

Arguments optionnels :

[-p] affiche la matrice résultat contenant uniquement les valeurs, **sans texte superflu**

[-t] affiche le temps d’exécution en millisecondes, **sans unité ni texte superflu**.

Par exemple, pour deux exemplaires de taille 4, la commande de la première ligne fournit la sortie suivante :



On y trouve d’abord la matrice de taille 16\*16 puis le temps d’exécution en ms.

Important : l’option -e doit pouvoir accepter des chemins absolus.

**Qualité du code**

|  | / 1 pt |
| --- | --- |

**Présentation générale**

|  | / 1 pt |
| --- | --- |

* Concision
* Qualité du français

**Pénalité retard**

| 0 |
| --- |

* -1 pt / journée de retard, arrondi vers le haut. Les TPs ne sont plus acceptés après 3 jours.