INF8775 – Analyse et conception d’algorithmes

TP1 – Hiver 2023

|  |  |
| --- | --- |
| **Nom, prénom, matricule des membres** | Dépelteau, Nicolas, 2083544  Turcotte, Alexandre, 2087684 |
| **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

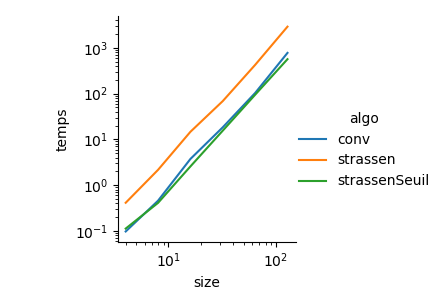
Présentation des résultats

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

Tableau des résultats

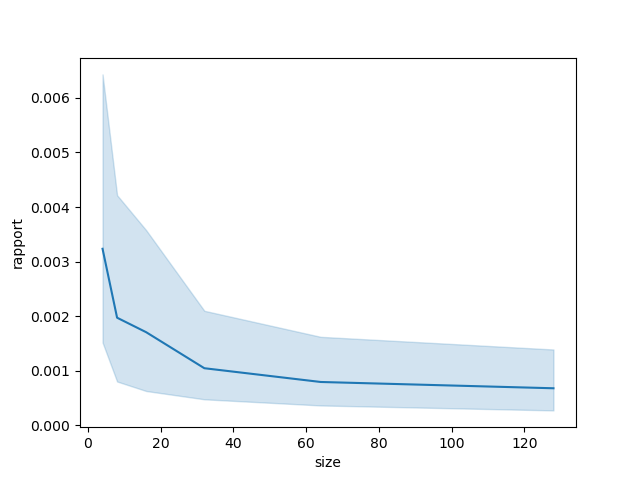
|  |  |  |
| --- | --- | --- |
| **Algorithme** | **Taille** | **Temps (ms)** |
| Conventionnelle | 4 | 0,0970 |
| Conventionnelle | 8 | 0,4570 |
| Conventionnelle | 16 | 3,7316 |
| Conventionnelle | 32 | 18,3589 |
| Conventionnelle | 64 | 103,1897 |
| Conventionnelle | 128 | 776,6825 |
| Strassen | 4 | 0,4117 |
| Strassen | 8 | 2,1591 |
| Strassen | 16 | 14,6437 |
| Strassen | 32 | 68,7046 |
| Strassen | 64 | 424,4527 |
| Strassen | 128 | 2908,9042 |
| Strassen avec seuil | 4 | 0,1123 |
| Strassen avec seuil | 8 | 0,4095 |
| Strassen avec seuil | 16 | 2,5625 |
| Strassen avec seuil | 32 | 15,5109 |
| Strassen avec seuil | 64 | 94,5369 |
| Strassen avec seuil | 128 | 566,1365 |

Tests de puissance

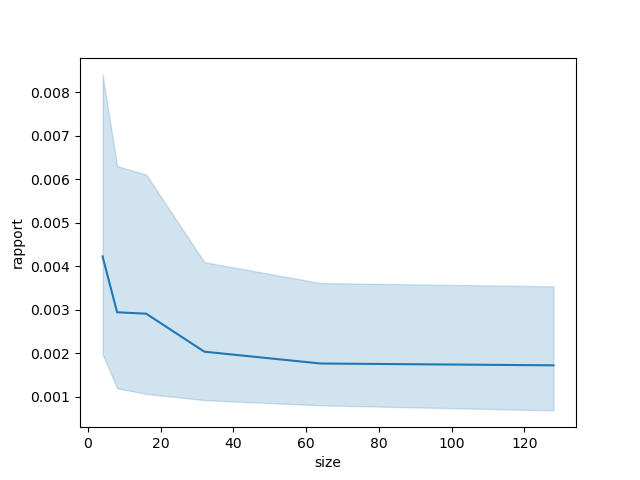
**

Test du rapport

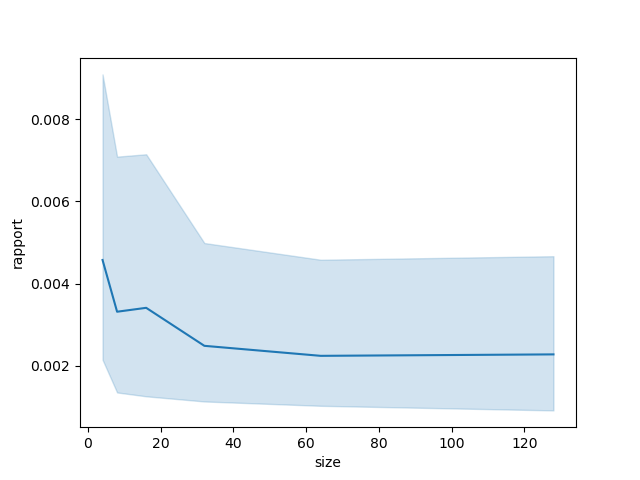
Graphique pour l’algorithme conventionnel :

**

Graphique pour l’algorithme de Strassen

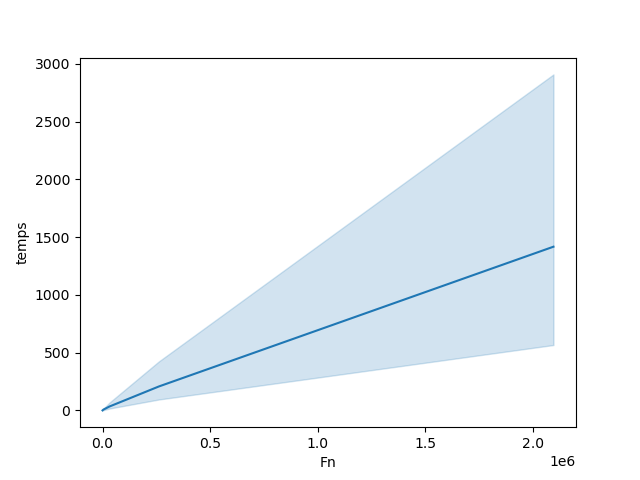


Graphique pour l’algorithme de Strassen avec un Seuil

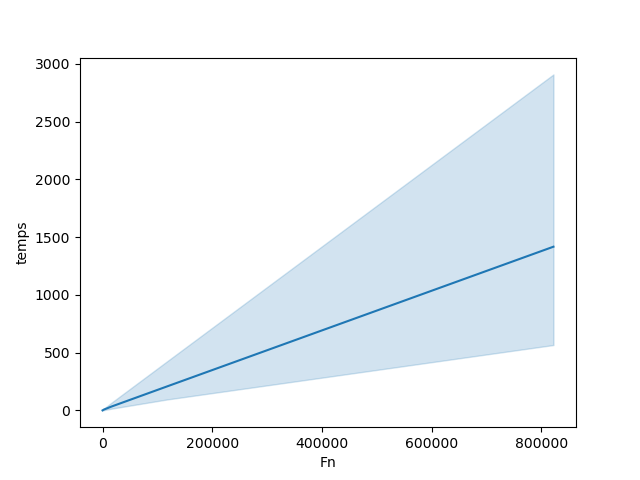


Test des constantes

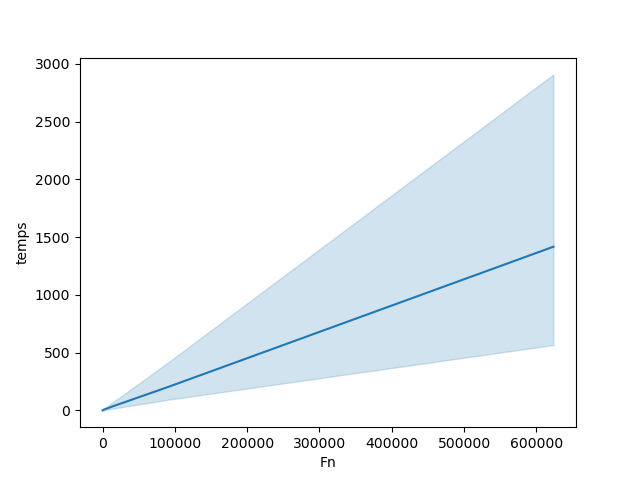
Graphique pour l’algorithme conventionnel :



Graphique pour l’algorithme de Strassen



Graphique pour l’algorithme de Strassen avec un Seuil



Analyse et discussion

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

Que pouvez-vous déduire du test de puissance ?

Nous pouvons déduire du test de puissance que nos trois algorithmes ont un temps polynomial. Effectivement, nos trois résultats suivent une droite, ce qui nous permet cette déduction. La pente pour les algorithmes conventionnel, Strassen et Strassen avec seuil sont respectivement de 2.588, 2.543 et de 2.504. Nous remarquons aussi que Strassen à une ordonnée à l’origine plus grande que les deux autres.

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

Conventionnel :

Strassen :

Strassen avec seuil :

Que pouvez-vous déduire du test du rapport ?

Pour le test du rapport, nous avons posé des hypothèses pour la puissance de « n », voici les hypothèses utilisées :

Conventionnel : 3

Strassen : 2.807

Strassen avec seuil : 2.75

Avec les graphiques des algorithmes respectifs, nous pouvons aisément déduire que nos hypothèses sont juste, car les trois algorithmes convergent tous vers une valeur. L’algorithme conventionnel converge environ vers une valeur de 0.0007, l’algorithme de Strassen converge environ vers 0.0018 et, finalement, l’algorithme de Strassen avec un seuil converge environ vers 0.0022.

Que pouvez-vous déduire du test des constantes ?

Le test des constantes nous permet aussi de vérifier nos hypothèses posées pour le test du rapport. Puisque les graphiques sont linéaires, nous pouvons déduire que nos hypothèses sont justes. De plus, nous pouvons observer trois pentes pour nos trois algorithmes (conventionnel, Strassen et Strassen avec seuil) de respectivement 0.00067, 0.00172 et 0.00227. Nous pouvons aussi constater que nos valeurs de pente sont environ égales à nos estimations de la convergence des tests du rapport.

Discutez de l’impact du seuil de récursivité.

Le seuil nous permet de passer à l’algorithme conventionnel pour de petites tailles de matrice, car Strassen n’est efficace que pour les matrices de grandes tailles. De plus, rajouter le seuil n’affecte pas la complexité théorique de l’algorithme, il ne fait que réduire le temps d’exécution en pratique.

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

Nous utiliserions l’algorithme conventionnel pour des matrices de petites tailles et il a l’avantage d’avoir une implémentation naïve, ce qui réduit son temps d’implémentation. Donc si nous avons des contrainte de temps dans le développement de l’algorithme, le conventionnel peut être une bonne option. Cet algorithme requiert aussi peu de mémoire dans un système limité en mémoire.

Nous utiliserions l’algorithme de Strassen pour des matrices plus grandes et le temps d’implémentation reste relativement faible malgré la subdivision de matrice. Cet algorithme devrait être utilisé dans des systèmes ayant beaucoup de mémoire, car il nécessite de créer des sous-matrices.

Nous utiliserions l’algorithme de Strassen avec un seuil pour un échantillon de matrice de toute taille. Cependant, il requiert d’implémenter les deux algorithmes (conventionnel et Strassen) et de calibrer le seuil, ce qui coûte plus de temps d’implémentation. Les contraintes de mémoire sont les mêmes que pour l’algorithme de Strassen.

Autres critères de correction

Respect de l’interface tp.sh

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

Utilisation :

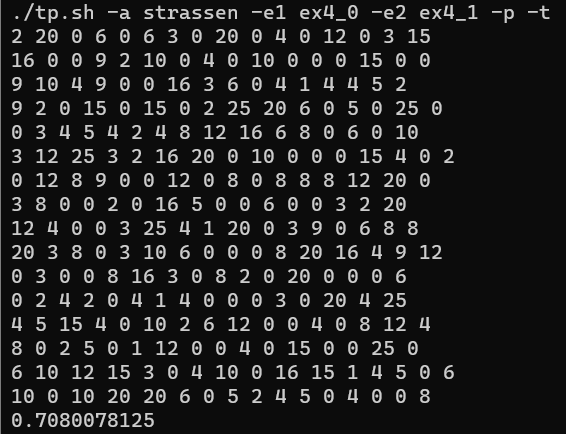
$ ./tp.sh -a {conv, strassen, strassenSeuil} -e1 PATH\_VERS\_EX\_1 -e2 PATH\_VERS\_EX\_2 [-p] [-t]e

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.