INF8775 – Analyse et conception d’algorithmes

TP1 – Hiver 2023

|  |  |
| --- | --- |
| **Nom, prénom, matricule des membres** | MOKRANI, Manel, 1990012  PERREAULT, Jérémy, 1903274 |
| **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

Dans le cadre du travail pratique, le temps d’exécution des exemplaires a été mesuré à l’aide de la librairie **time**. Ces valeurs peuvent être observées à la figure 1.

Le tableau suivant présente ces données plus clairement :

|  |  |  |  |
| --- | --- | --- | --- |
| Taille des exemplaires | Temps moyen () | | |
| Conventionnel | Strassen | Strassen seuil |
| 16 | 0.003884077 | 0.017350721 | 0.017644787 |
| 32 | 0.028643131 | 0.121838045 | 0.121400404 |
| 64 | 0.226339102 | 0.840823841 | 0.817908525 |
| 128 | 1.67718792 | 5.812413168 | 5.738332462 |
| 256 | 13.05098381 | 40.53863888 | 13.0688004 |

Tableau 1. Mesure du temps d’exécution des exemplaires

Tests de puissance

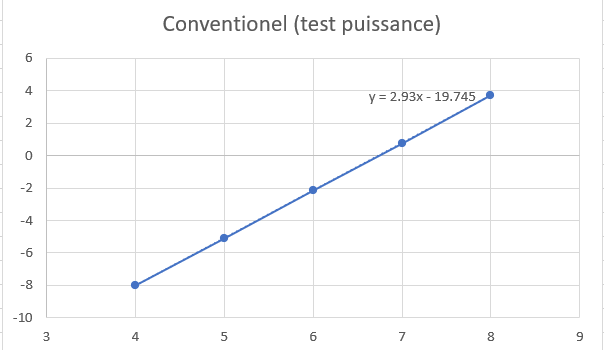


Figure 1: test de puissance de l’algorithme conventionnel (log-log)

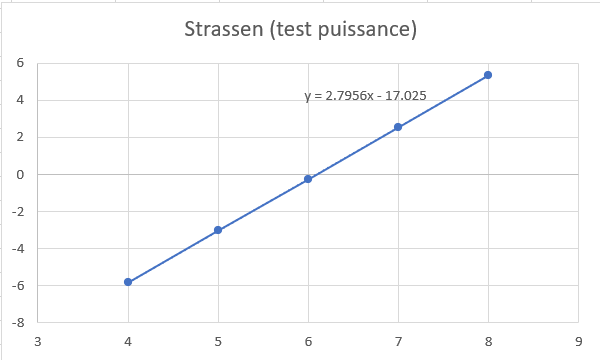


Figure 2: test de puissance de Strassen (log-log)

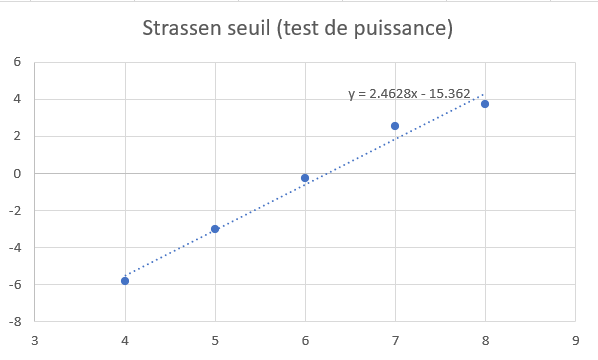


Figure 3 : test de puissance Strassen seuil (log-log)

Test du rapport

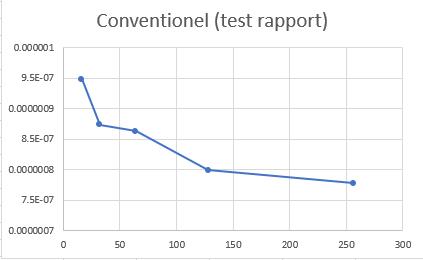


Figure 4: test de rapport de l’algorithme conventionnel (x-y/f(x))

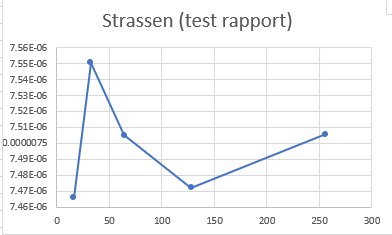


Figure 5: test de rapport de Strassen (x-y/f(x))

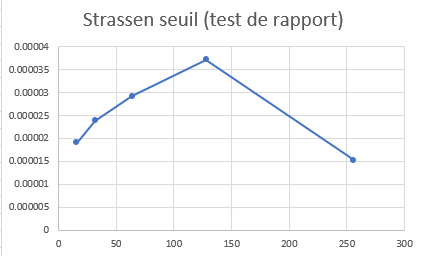


Figure 6 : test de rapport Strassen seuil (x-y/f(x))

Test des constantes

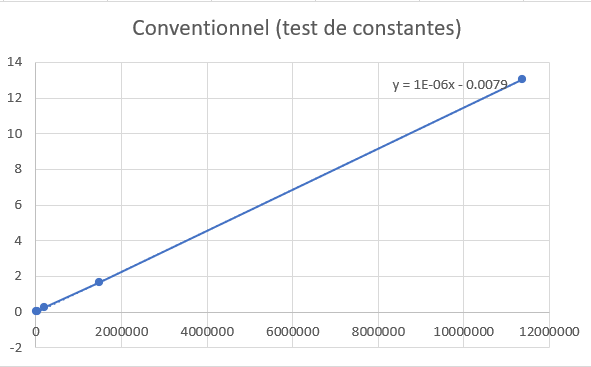


Figure 7: test de constantes de l’algorithme conventionnel (y, f(x))

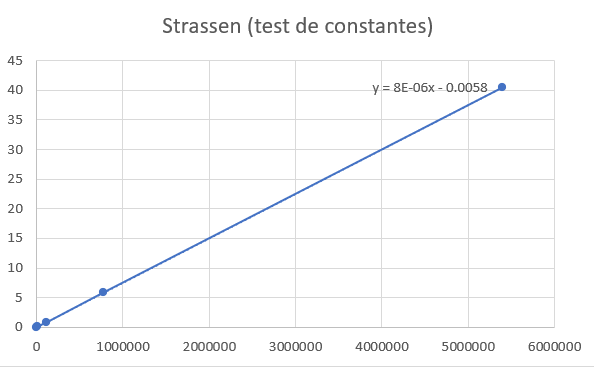


Figure 8: test de constante de Strassen (y, f(x))

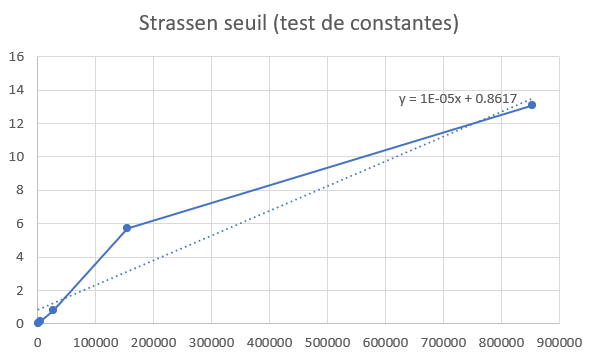


Figure 6 : test de constantes Strassen seuil (y, f(x))

Analyse et discussion

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

Que pouvez-vous déduire du test de puissance ?

En théorie, le test de puissance permet, pour un algorithme quelconque, de donner une idée générale du taux de croissance des ressources consommés en fonction de la taille des exemplaires à résoudre.

Dans le cas présent, il est possible de constater que chacun des algorithmes (conventionnel, Strassen, Strassen avec seuil) a une consommation (en temps) polynomial, puisque le résultat du test de puissance donne une droite dans chacun des cas. Pour l’algorithme conventionnel nous obtenons la droite , pour l’algorithme Strassen , et pour l’algorithme Strassen avec seuil .

Étant donné que ces trois algorithmes ont un taux de croissance polynomial, il est possible d’appliquer la régression linéaire suivante :

Où représente la base de l’échelle logarithmique, la pente et la valeur initiale.

Nous pouvons donc émettre l’hypothèse que la croissance de la consommation (en temps) de l’algorithme conventionnel augmentera en fonction de , celle de Strassen en fonction de , et celle de Strassen avec seuil en fonction de .

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

À partir des résultats obtenus au test de puissance, il est possible de déduire la consommation théorique du temps de calculs pour les algorithmes. La notation asymptotique de ceux-ci sont présentés au tableau suivant en comparaison avec des valeurs théoriques citées de la littérature (Wikipédia, 2022) :

|  |  |  |
| --- | --- | --- |
| Algorithme | Complexité théorique | Complexité expérimentale |
| Conventionnel |  |  |
| Strassen |  |  |
| Strassen avec seuil |  |  |

Tableau 2. Complexité théorique et expérimentale des différents algorithmes.

Il est possible de constater que les valeurs obtenues expérimentalement concordent avec les valeurs théoriques. Cela valide donc les résultats.

Il est à noter que la complexité théorique de l’algorithme de Strassen avec seuil n’a pas pu être trouvée. Voici la référence bibliographique de notre recherche :

Wikipédia. (2022). *Algorithme de Strassen*. Wikipédia. <https://fr.wikipedia.org/wiki/Algorithme_de_Strassen>

Que pouvez-vous déduire du test du rapport ?

En général, le test du rapport est utilisé lorsqu’on a déjà une bonne idée du taux de croissance d’un algorithme. Celui-ci est souvent déterminé à partir de l’analyse théorique. Le test du rapport permet d’analyser la relation .

Au cours de ce laboratoire, les observations effectuées ont permis de réaliser que l’algorithme conventionnel et celui de Strassen tous deux convergent vers 0. Ce qui veux dire que l’on a fait une surestimation. Quant à l’algorithme de Strassen avec seuil, la fonction converge vers une constante , ce qui veut dire que l’estimation est valide et que la complexité peut être modélisée par l’équation suivante :

Que pouvez-vous déduire du test des constantes ?

Finalement, le test des constantes, lorsqu’appliqué, se base lui-aussi sur une hypothèse préétablie du taux de croissance . Or, celui-ci permet d’observer la relation et permet d’obtenir une régression linéaire modélisées par l’équation :

À partir des résultats obtenus, il est possible de déduire une constante multiplicative et un coût fixe pour estimer la consommation de ressource de chaque algorithme. Les équations suivantes le démontrent :

Discutez de l’impact du seuil de récursivité.

Le choix du seuil de récursivité est très important. En effet, un seuil trop petit peut faire en sorte que l’algorithme prennent beaucoup de temps à s’exécuter en raison des nombreux appels récursifs qu’un petit seuil engendrerait. À l’opposé, un seuil trop grand ferait en sorte que l’on perdrait de l’efficacité de l’algorithme puisque l’on reviendrait à faire des multiplications conventionnelles de trop grandes matrices. Il nous faut donc trouver un juste milieu pour tirer avantage de l’algorithme Strassen avec seuil. Pour ce faire, nous pouvons recourir à la méthode essaie/erreur afin de trouver le seuil qui convient aux différentes tailles de nos matrices. Dans notre cas, nous avons convenue qu’un seuil de 256 serait convenables pour nos exemplaires qui ont des tailles entre 4 et 256.

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

En se basant sur notre tableau des résultats, nous pouvons constater que l’algorithme conventionnel est le meilleur des trois lorsqu’il est question de matrice de petite et de moyenne taille (taille <256).

L’algorithmes Strassen, quant à lui, ne donne jamais de résultats avantageux. C’est, en effet, le pire des deux algorithmes quant à la consommation de ressources en temps. Il est à noter qu’il soit fort probable que la taille des matrices choisies pour notre expérimentation ne soit pas assez grande pour constater l’utilité de l’algorithme Strassen

Enfin, l’algorithme Strassen avec seuil semble avoir un temps moyen se rapprochant de celui de l'algorithme conventionnel aux alentours des exemplaire de taille 256. Passé ce seuil, il semble consommer moins de ressources en temps que l’algorithme conventionnel.

En conclusion, nous n'utiliserons jamais l’algorithmes Strassen. Cependant, il est intéressant d’utiliser l’algorithme conventionnel pour les matrices de taille 256 et moins. Pour des matrices de taille plus grande l’algorithme Strassen avec seuil (seuil = 256) serait à privilégier.

Avec des échantillons de plus grandes tailles, il serait possible d’obtenir des estimés plus précis sur la consommation de chaque algorithme.

Autres critères de correction

Respect de l’interface tp.sh

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

Utilisation :

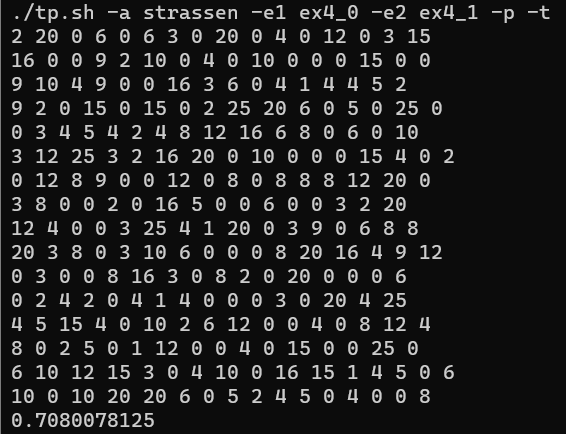
$ ./tp.sh -a {conv, strassen, strassenSeuil} -e1 PATH\_VERS\_EX\_1 -e2 PATH\_VERS\_EX\_2 [-p] [-t]

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.