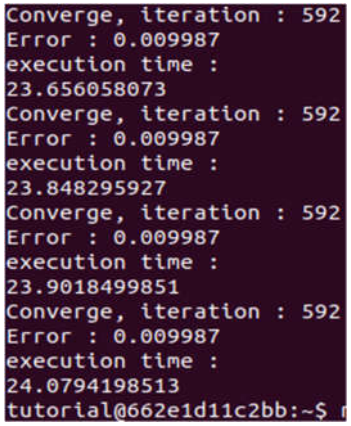
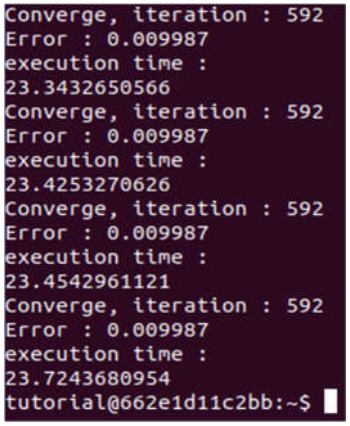
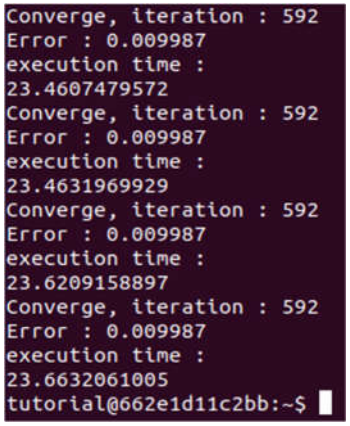
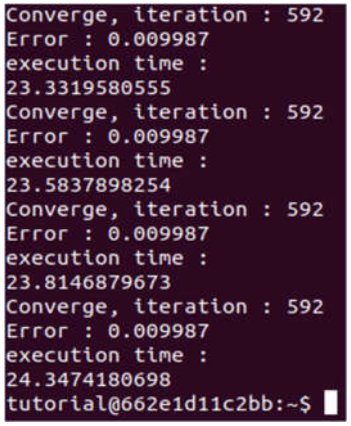
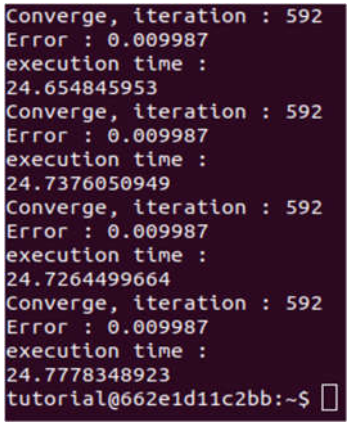
Compte-rendu
TP - Parallélisation en environnement MPI

Guillaume COLLET

Le code source est récupérable à cette adresse : **https://github.com/LomigFR/M2_CCN_MPS_TP_PARALLELISATION_MPI**
Dans la suite de ce compte-rendu, mon code sera désigné par « **code avec parallélisation** » et celui fourni en classe (sans parallélisation) sera désigné par « **code témoin** ».
Cinq mesures de temps ont été réalisées pour le code avec parallélisation ainsi que pour le code témoin, avec une **matrice de 64x64** (les deux paddings inclus) dans chaque cas.
Ces 10 mesures ont été réalisées dans les mêmes conditions à savoir sur un laptop **Dell Latitude 5580** (branché sur secteur) prêté par l'établissement, et dont les spécifications techniques sont :

CPU : Intel Core i3 7100U @ 2.40 GHz x 4
RAM : 8 Go
Carte graphique : Intel HD Graphics 620 (Kaby Lake GT2)
SSD : 128 Go
OS : Ubuntu 18.04.4 LTS (64 bits)
Environnement : Gnome 3.28.2

Mesure_id	Résultats obtenus avec parallélisation	Temps moyens de calculs par série de mesures (arrondis au centième)
1	('Total normalized difference:', 0.009991822186609645) ('Execution time for node number', 3, ' :', 12.032201766967773, ' seconds') ('Execution time for node number', 1, ' :', 12.031753063201904, ' seconds') ('Execution time for node number', 2, ' :', 12.025652885437012, ' seconds') ('Dimensions of the final matrix (includes previous ones-padding):', (62, 62)) ('Number of iterations :', 883) ('Execution time for node number', 0, ' :', 12.038275957107544, ' seconds') tutorial@662e1d11c2bb:~$	mean_duration_1 = (12.03 + 12.03 + 12.02 + 12.04) / 4 ➔ **12.03 secondes**
2	('Total normalized difference:', 0.009991822186609645) ('Execution time for node number', 2, ' :', 12.58613395690918, ' seconds') ('Execution time for node number', 1, ' :', 12.589017152786255, ' seconds') ('Execution time for node number', 3, ' :', 12.588917970657349, ' seconds') ('Dimensions of the final matrix (includes previous ones-padding):', (62, 62)) ('Number of iterations :', 883) ('Execution time for node number', 0, ' :', 12.59059190750122, ' seconds') tutorial@662e1d11c2bb:~$	mean_duration_2 = (12.59 + 12.59 + 12.59 + 12.59) / 4 ➔ **12.59 secondes**
3	('Total normalized difference:', 0.009991822186609645) ('Execution time for node number', 1, ' :', 13.552447080612183, ' seconds') ('Execution time for node number', 2, ' :', 13.553853988647461, ' seconds') ('Dimensions of the final matrix (includes previous ones-padding):', (62, 62)) ('Number of iterations :', 883) ('Execution time for node number', 0, ' :', 13.555577993392944, ' seconds') ('Execution time for node number', 3, ' :', 13.555011987686157, ' seconds') tutorial@662e1d11c2bb:~$	mean_duration_3 = (13.55 + 13.55 + 13.56 + 13.55) / 4 ➔ **13.55 secondes**
4	('Total normalized difference:', 0.009991822186609645) ('Execution time for node number', 2, ' :', 13.012071132659912, ' seconds') ('Execution time for node number', 1, ' :', 13.013383150100708, ' seconds') ('Execution time for node number', 3, ' :', 13.024245023727417, ' seconds') ('Dimensions of the final matrix (includes previous ones-padding):', (62, 62)) ('Number of iterations :', 883) ('Execution time for node number', 0, ' :', 13.024944067001343, ' seconds') tutorial@662e1d11c2bb:~$	mean_duration_4 = (13.01 + 13.01 + 13.02 + 13.02) / 4 ➔ **13.01 secondes**
5	('Total normalized difference:', 0.009991822186609645) ('Execution time for node number', 1, ' :', 13.041556119918823, ' seconds') ('Execution time for node number', 2, ' :', 13.038604021072388, ' seconds') ('Execution time for node number', 3, ' :', 13.044471025466919, ' seconds') ('Dimensions of the final matrix (includes previous ones-padding):', (62, 62)) ('Number of iterations :', 883) ('Execution time for node number', 0, ' :', 12.956557989120483, ' seconds') tutorial@662e1d11c2bb:~$	mean_duration_5 = (13.04 + 13.04 + 13.04 + 12.96) / 4 ➔ **13.02 secondes**
Temps de calculs moyens par nœud		mean_duration_node_1 : 12.84 s mean_duration_node_2 : 12.84 s mean_duration_node_3 : 12.84 s mean_duration_node_4 : 12.83 s
Moyenne globale (sur les 20 valeurs) : **(mean_duration_1 + mean_duration_2 + mean_duration_3 + mean_duration_4 + mean_duration_5) / 5 =** **12.84 secondes**		

Mesure_id	1	2	3	4	5
Résultats obtenus sans parallélisation (code témoin)					
Temps moyens de calculs par série de mesures **(arrondis au centième)**	mean_duration_1 = (**23.66** + **23.85** + **23.90** + **24.08**) / 4 ➔ **23.87 secondes**	mean_duration_2 = (**23.34** + **23.42** + **23.45** + **23.72**) / 4 ➔ **23.48 secondes**	mean_duration_3 = (**23.46** + **23.46** + **23.62** + **23.66**) / 4 ➔ **23.55 secondes**	mean_duration_4 = (**23.33** + **23.58** + **23.81** + **24.35**) / 4 ➔ **23.77 secondes**	mean_duration_5 = (**24.65** + **24.74** + **24.73** + **24.78**) / 4 ➔ **24.72 secondes**

Moyenne globale (sur les 20 valeurs) :

(mean_duration_1 + mean_duration_2 + mean_duration_3 + mean_duration_4 + mean_duration_5) / 5

= 23.88 secondes

Observations.

- Les temps de calculs **avec parallélisation** sont bien inférieurs à ceux obtenus avec le code témoin (**sans parallélisation**).

- Le gain de temps ici est de l'ordre de **46%**.

- Dans le cas de la parallélisation, on peut par ailleurs observer que les temps de calculs moyens par nœud sont très proches pour ne pas dire identiques.

- Toutefois, le nombre d'itérations de calculs avec parallélisation est supérieur au nombre d'itérations effectuées par le code témoin (**+49%** environ).

Analyses, interprétations.

N'ayant pas d'idée de l'ordre de grandeur du gain de temps ni du nombre d'itérations que l'on pouvait attendre en parallélisant ce travail de convolution, je ne sais pas si ces résultats sont significatifs et sont donc difficiles à analyser pour moi.
Un retour à ce sujet m'intéresserait, si cela est possible.

Remarques diverses.

- Dans le code avec parallélisation, le **chronométrage ne démarre qu'après le chargement des différentes fonctions** proposées, afin d'être au plus proche des conditions de mesures du temps de calcul présentées dans le code témoin.

- Lors des mesures, les **affichages de matrices ont été désactivés** (lignes de code correspondantes commentées). Pour le code avec parallélisation, il est possible d'afficher la matrice finale en **décommentant la ligne 114**.

- Le code avec parallélisation proposé ici, fonctionne pour une **taille de matrice multiple de 4**. Les mesures précédentes ont été faites à partir d'une **matrice de 64x64** (padding de 1 et padding de 0 inclus) que ce soit pour le code témoin ou pour le code avec parallélisation.
  Les méthodes de construction des matrices étant différentes dans les deux cas (bien qu'arrivant au même résultat), ceci se traduit par **Max_N = 62** dans le code témoin et **N = 60** dans le code avec parallélisation (cf. fonction `create_square_zero_matrix(N)`).

- Dans le code avec parallélisation, **deux fonctions sont proposées en vue de calculer la différence normalisée** sur laquelle s'appuie le code pour arrêter la convolution. Ceci a été fait uniquement par volonté personnelle en vue de comparer d'éventuelles différences de performances entre les deux méthodes de calculs suivantes :

  - **Méthode de calcul 1** (fournie dans le code témoin et aussi utilisée dans le code avec parallélisation) :

$$diff\_norm = \sqrt{(a-b)^2}$$

  - **Méthode de calcul 2** (proposée en plus dans le code avec parallélisation → cf. lignes commentées) :

$$diff\_norm = |a-b|$$

  Quelques mesures (non présentées ici) ont été réalisées avec chacune de ces deux méthodes que ce soit pour le code témoin ou pour le code avec parallélisation.

  Observations :

  - La **méthode de calcul 1 permet de gagner de l'ordre de 2 à 3 secondes** de temps de calculs moyen global dans le cas de la parallélisation vs la méthode calcul 2.

  - La **méthode de calcul 2 permet de gagner de l'ordre de 4 secondes** de temps de calculs moyen global dans le cas sans parallélisation vs la méthode calcul 1.