Question 2

Dans le cas décroissant, on commence par aditionner/soustraire les nombres les plus petits contrairement au cas inverse où l'arrondi induit une imprécision.

Question 3

Soit un thread calcule un bloc continu de la série, soit on divise la serie de taille n en m parties et chaque thread calcule l'élément num_thread de chaque partie.

Question 6

Sur CPU (version descendante) on a un temps d'exécution de 0.95s avec un résultat de 0.693137 alors que sur GPU on obtient un temps d'exécution de 0.189s avec un résultat de 0.693147.
En dédiant le calcul au GPU, on obtient une vitesse d'exécution plus rapide car le processeur du GPU est dédié "uniquement" à cette tâche (contrairement au traitement via CPU). Le GPU est d'autant plus rapide qu'il effectue plusieurs calculs en parallèle.
En utilisant le calcul par rang, on obtient un temps d'exécution de 0.216s et un résultat de 0.703252. Cette méthode est moins précise pour les mêmes raisons que présentées en question 2.

Question 7

Quand on utilise un nombre de threads trop petit, on sous-exploite nos capacités de calcul (augmentation du temps d'exécution). Dans le cas contraire, on atteind le "seuil de parallèlisme" de notre carte et les threads en trop n'apportent rien voir ralentissent l'exécution à cause de la synchronisation.
Par conséquent les valeurs par défaut conviennent à un temps d'exécution optimal (threads/bloc = 128 et nombre de blocs = 8).

Question 8

On peut faire une réduction des valeurs retournées dans un bloc à l'aide d'une mémoire partagée entre les threads et ainsi les additionner coté GPU afin de ne retourner qu'une valeur par bloc.