RAPPORT – EXERCICE 3

Microprocessor Architectures

Mourad AKANDOUCH
Issam HAJJI
Mounir HAFIF

10 MAI 2017

Partitionnement du tableau de bytes

Pour découper le tableau de bytes dans le but de le répartir entre les threads de la manière la plus efficace possible, nous avons pensé à diviser le tableau en parts égales. Le tableau de pixels pouvant se représenter en une dimension, nous avons attribuer les N premiers pixels au premier thread, les N suivants au second, etc. où $N = \frac{1024^2}{nombre\ de\ threads}$

Cette manière de répartir les pixels nous a semblé optimale niveau performances car les pixels attribués à chaque thread sont contigus en mémoire (*Mais comme le tableau se trouve en RAM, il s'agit d'un faux argument je pense*).

Comparaison avec deux, quatre et huit threads

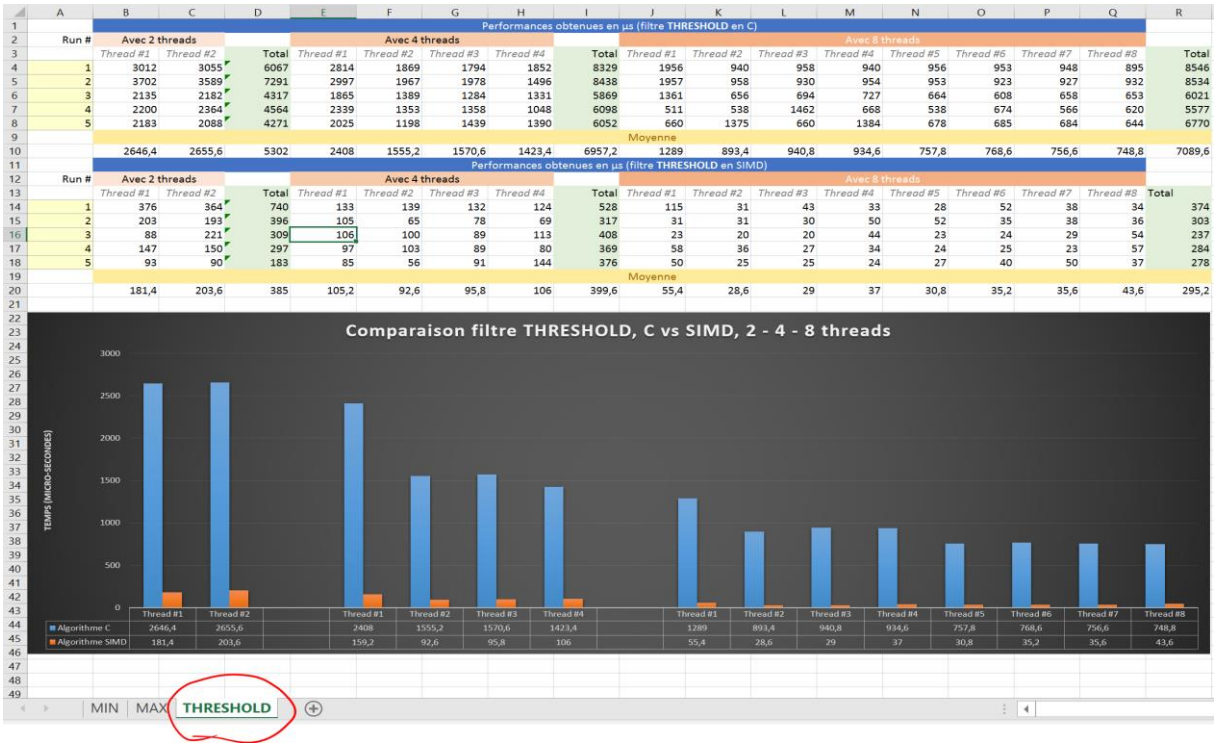
Filtre THRESHOLD

Nous avons constaté une nette amélioration en fonction du nombre de threads créés. Les performances individuelles de chaque thread étaient environs deux fois meilleures lorsque nous doublions le nombre de threads. Cependant, le temps total (*somme des performances de chaque thread*) se dégradait en fonction du nombre de threads créés.

Avec deux threads, nous avons calculé un temps moyen de 5302 µs ; avec 4 threads, il était de 6957 µs, et avec 8, le temps moyen était de 7089 µs. Apparemment, la différence de temps devient moindre avec le nombre de threads créés. Le comportement semble être le même, tant en C qu'en SIMD.

Il est également à noter que nos hypothèses dépendent de 5 exécutions du programme et peuvent donc être biaisées par l'ordonnancement. Nous devrions tenter davantage d'exécutions pour émettre de meilleures conclusions. Vous pouvez trouver en annexe un fichier Excel répertoriant le benchmark du filtre THRESHOLD (sous l'onglet THRESHOLD).

Voici un aperçu ci-dessous :



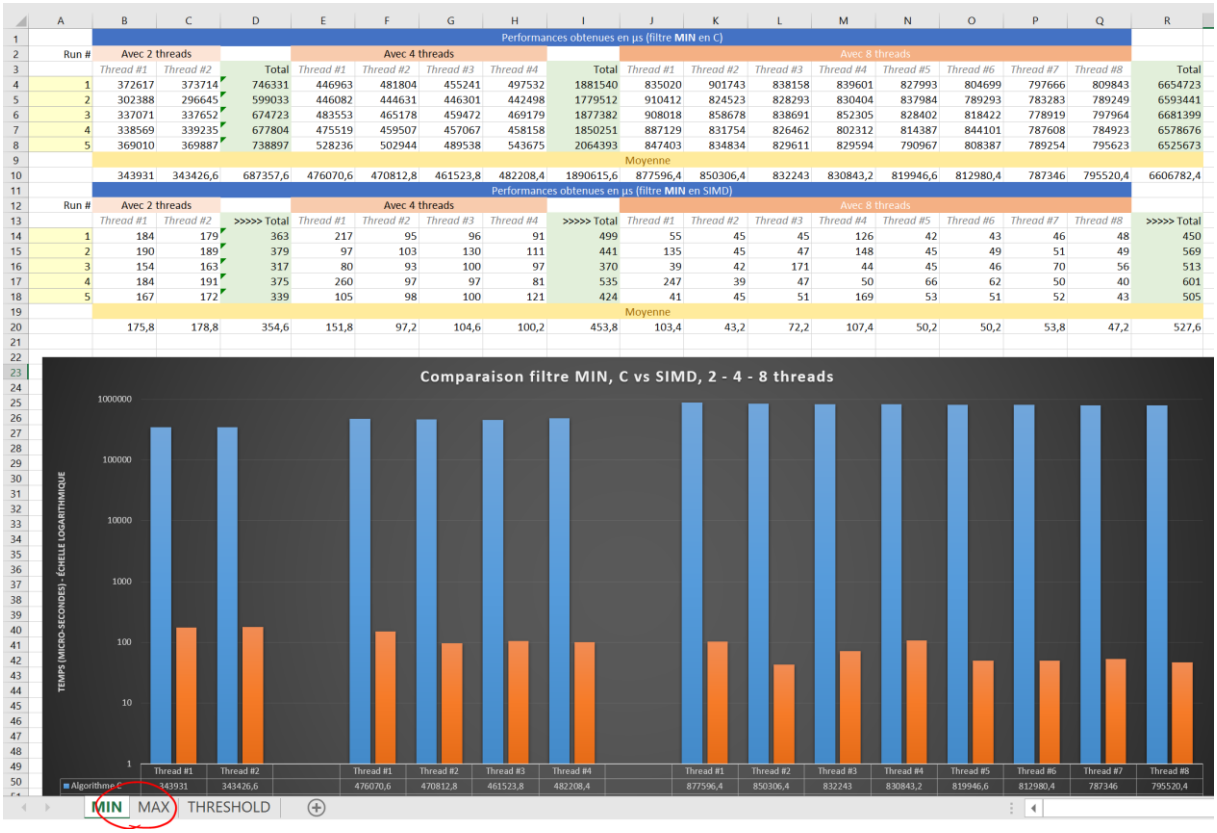
Filtre MIN et MAX

Contrairement à l'intuition qui pourrait s'en dégager au vue des observations constatées lors de la mesure des performances des filtres MIN et MAX, nous avons observé des résultats surprenants et certainement non liés à l'ordonnancement.

En C, nous avons observé une nette dégression des performances individuelles des threads lorsque ceux-ci sont en plus grand nombre. En moyenne, un thread prenait deux fois plus de temps à s'exécuter lorsque leur nombre double. Par exemple, en moyenne, pour deux threads, le temps moyen d'exécution d'un thread était de 300 000 µs. Pour quatre threads en parallèle, le temps d'exécution moyen d'un thread passait à 470 000 µs ! Et pour 8 thread, le temps moyen d'un thread explosait et passait à 850 000 µs ! (Ces temps correspondent au filtre MAX. Le temps est quasiment le même pour le filtre MIN)

En SIMD, le résultat est contraire, le temps individuel des threads diminuait lorsque leur nombre augmentait. Cependant, le temps total moyen augmentait (tout comme en C d'ailleurs).

Voici ci-dessous un aperçu des résultats (trouvables dans le fichier Excel en annexe dans le dossier, sous les onglets MIN et MAX) :



A noter que sur les graphes représentant les performances des filtres MIN et MAX, l'échelle de temps est logarithmique, sinon nous n'aurions pas pu voir les barres orangées correspondant aux performances en SIMD. *(Nous pouvons voir que les barres bleues augmentent drastiquement (contexte logarithmique) lorsqu'il y a davantage de threads, et les barres orangées diminuent globalement.)*