Part 0: Discover your GPU

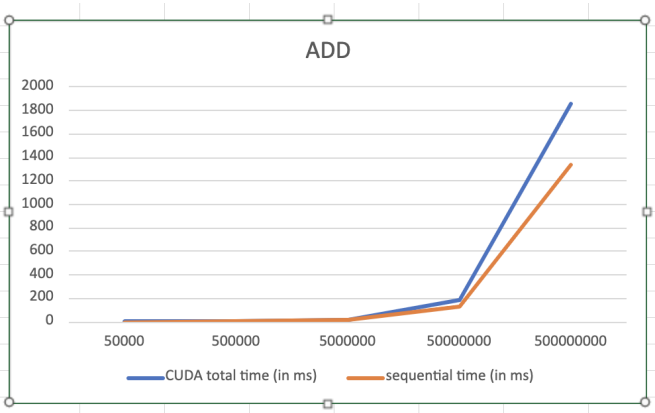
```
$ ./which-device

Device Name:
NVIDIA A100-PCIE-40GB
Total global memory:    42358472704
Shared memory / SM:     49152
Registers / SM: 65536
max threads/blocks:     1024
max threads in each dimension:  1024, 1024, 1024
max blocks in each dimension:   2147483647, 65535, 65535
Nb of multiproc on device:      10865535
```

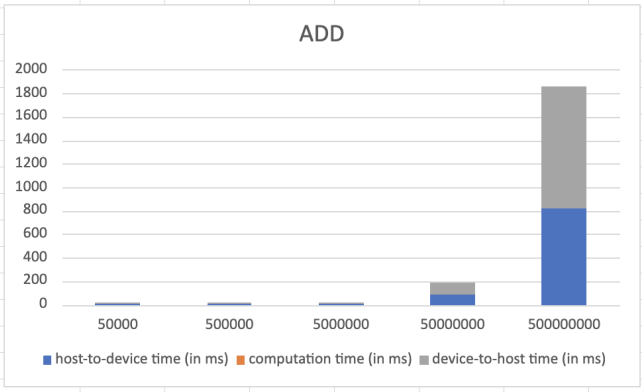
memory size, block and thread number, etc. Nous utilisons le service distant boole. Nous pouvons voir que nous possédons 10865535 Streaming Multiprocessors sur notre carte NVIDIA A100-PCIE-40GB. Chaque SM possède une mémoire partagé de
49152 octets et 65536 registres. Notre carte NVIDIA possède une mémoire globale de 42Go

Part I: Vector Addition

operations	block used	threads used	host-to-device time (in ms)	computation time (in ms)	device-to-host time (in ms)	Transfert time
50000	49	1024	1,446	0,021	0,159	1,605
500000	489	1024	0,957	0,025	1,306	2,263
5000000	4883	1024	8,648	0,024	11,136	19,784
50000000	48829	1024	83,240997	0,025	104,013	187,253997
500000000	488282	1024	822,393982	0,034	1036,791992	1859,185974

Nous pouvons clairement voir que le temps de transfert est énorme comparait au temps de calcul, en fait, CUDA arrive bien a paralléliser les opérations simples, cependant, pour passer de host au device et du device au host, nous perdons beaucoup plus de temps que si nous realisons l'opération en séquentiel.

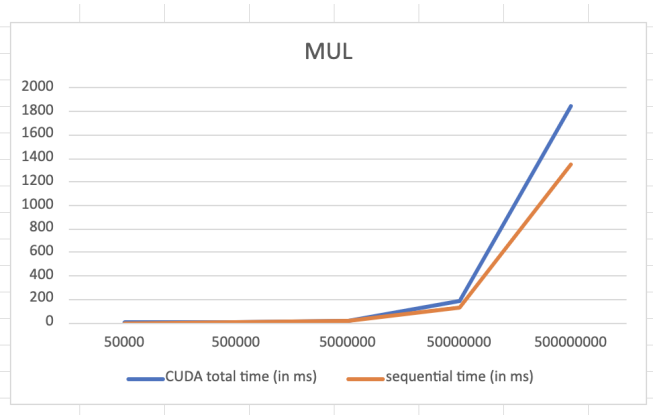


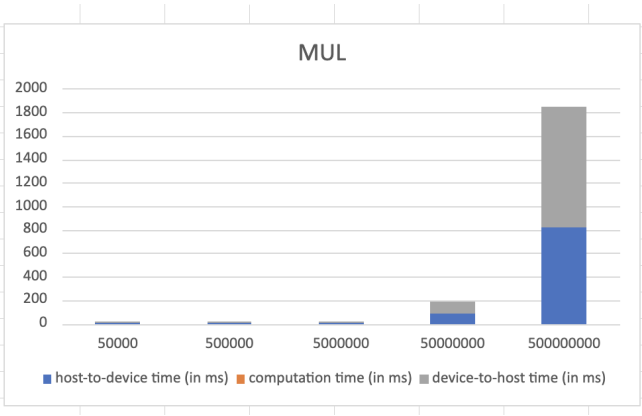


Le graphique nous le montre bien, nous ne voyons meme pas le temps d'éxécution, tout le temps est pris par le transfert.

Part II: Vector Multiplication

operations	block used	threads used	host-to-device time (in ms)	computation time (in ms)	device-to-host time (in ms)	Transfert time
50000	49	1024	1,575	0,039	0,324	1,899
500000	489	1024	0,991	0,024	1,372	2,363
5000000	4883	1024	8,473	0,018	10,738	19,211
50000000	48829	1024	84,050003	0,039	104,473999	188,524002
500000000	488282	1024	821,044006	0,042	1025,557983	1846,601989

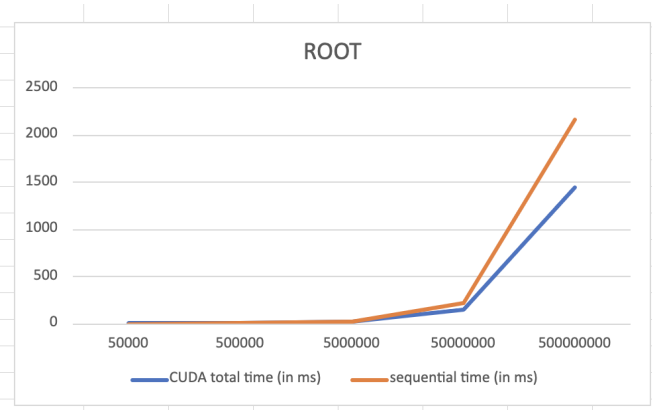


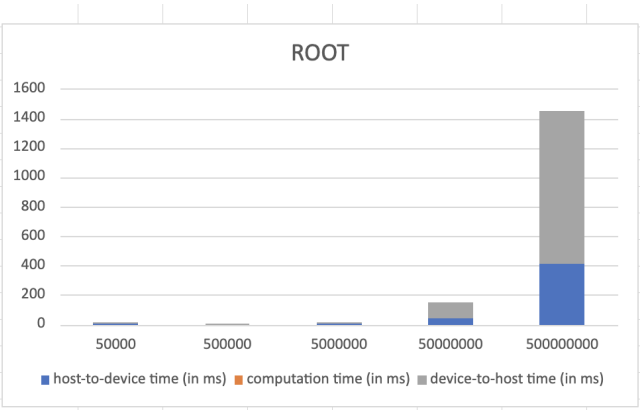


A nouveau, malgré le fait que l'opération soit plus compliqué, nous sommes toujours face a des durée de transfert beaucoup plus importantes que le temps de calcul en séquentiel. Nous pouvons donc commencer a conclure (mais nous y reviendrons à la suite) que globalement, pour des opérations simples, il n'est pas toujours intéressant de passer sur le GPU

Part III: Vector Square Root

operations	block used	threads used	host-to-device time (in ms)	computation time (in ms)	device-to-host time (in ms)	Transfert time
50000	49	1024	1,406	0,015	0,15	1,556
500000	489	1024	0,431	0,018	1,235	1,666
5000000	4883	1024	4,229	0,02	11,352	15,581
50000000	48829	1024	41,759998	0,025	104,080002	145,84
500000000	488282	1024	415,039001	0,027	1034,767944	1449,806945



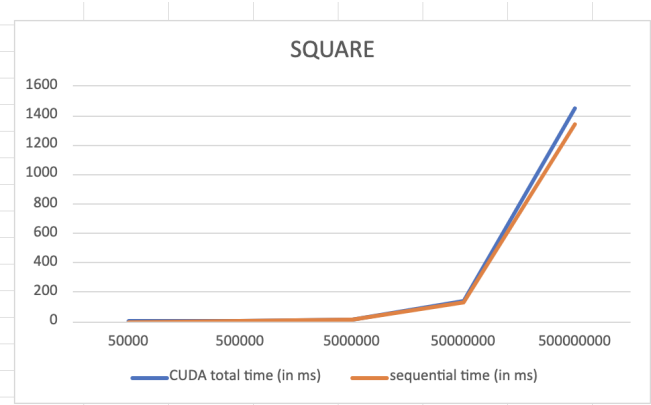


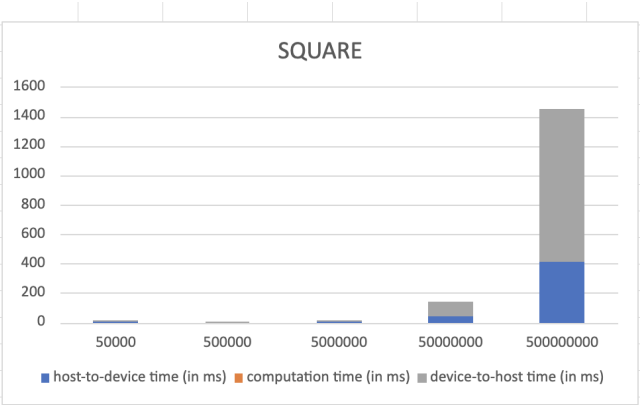
Ici, nous avons compléxifier l'opération entre les 2 matrices. En effet, la fonction sqrt() demande beaucoup plus de ressource qu'une simple addition ou multiplication (nous devons faire plusieurs opérations a la suite pour obtenir le sqrt - selon les implémentations). Et ici, nous battons le calcul séquentiel. Malgré les temps de transfert importants, le fait de paralléliser les opérations plutot que de les faire simultanément nous fait gagner du temps. De plus, la raison principale pour laquelle nous battons l'implémentation séquentielle, est que nous envoyons que nous transferons plus que 2 array contre 3 auparavant

Part IV: Vector Square

operations	block used	threads used	host-to-device time (in ms)	computation time (in ms)	device-to-host time (in ms)	Transfert time
50000	49	1024	1,326	0,017	0,154	1,48
500000	489	1024	0,464	0,019	1,253	1,717

operations	block used	threads used	host-to-device time (in ms)	computation time (in ms)	device-to-host time (in ms)	Transfert time
5000000	4883	1024	4,303	0,026	11,201	15,504
50000000	48829	1024	41,75	0,023	102,962997	144,712997
500000000	488282	1024	415,825989	0,028	1030,253052	1446,079041





Ici nous avons diminué la compléxité de calcul mais nous avons conservé l'envoie que de un seul array. Nous sommes a peu près sur les mêmes durées de traitement, toujours avec un long temps de transfert.

Conclusion

Nous pouvons conclure que le travail en parallèle via GPU est très intéressant si nous avons des calculs complexes, nottament du calcul graphique qui est fortement matriciel. Cependant, il y a une conséquence, le temps de transfert du GPU au CPU et inversement, et c'est la que nous détruisons toute la performance gagnée. Il faut donc être vigilant, il peut être très intéressant d'utiliser le parallélisme énorme que notre GPU nous propose, mais il faut optimiser les transfert pour ne transferer que le minimum, afin de garder des performances intéressantes. DOnc pour résumer, si on a des calculs importants, avec peu de transfert de matrice, l'utilisation du GPU est interessante. Sinon, il faut tester différentes implémentations De plus, CUDA via les fichier .cu permet une implémentation assez facile et rapide, et l'exploitage du GPU de manière vraiment très simple, même si il faut sqitcher du paradigme séquentiel au paradigme parallèle.