AVIGNON UNIVERSITÉ

Rapport TP3 – Programmation parallèle : GPU

Sarra Bensafi

20 novembre 2021

Master 1
Master Intelligence Artificielle
UE Programmation Parallèle

Responsables
Mickael Rouvier

UFR
SCIENCES
TECHNOLOGIES
SANTÉ

AVIGNON UNIVERSITÉ
CERI

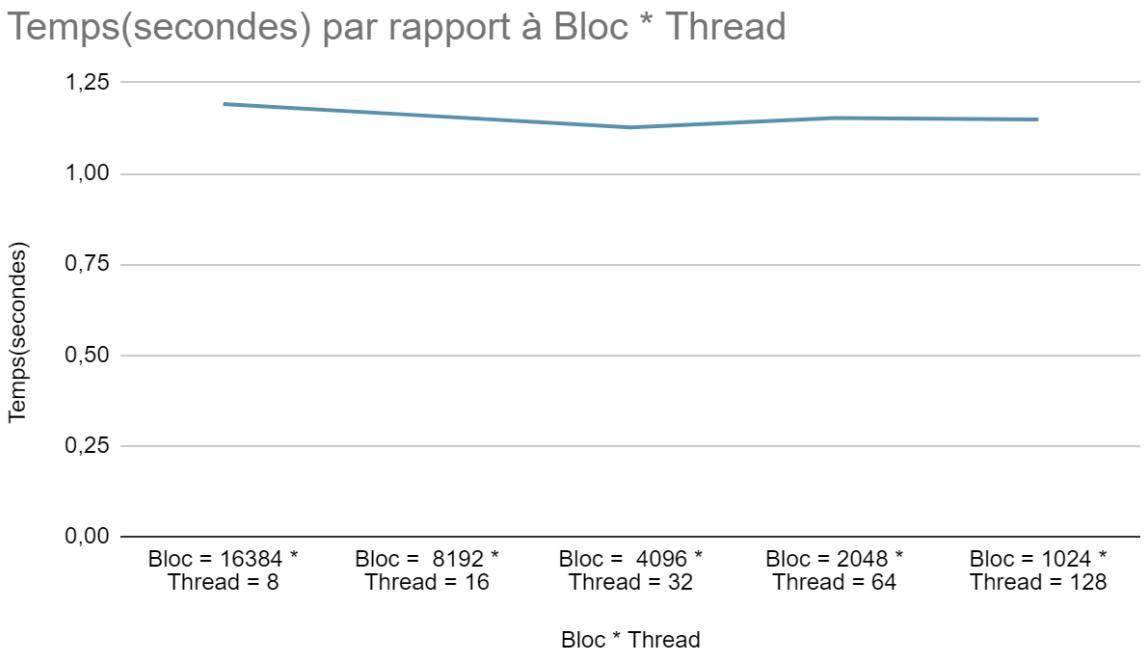
CENTRE
D'**ENSEIGNEMENT**
ET DE **RECHERCHE**
EN **INFORMATIQUE**
ceri.univ-avignon.fr

Exercice 1 Addition Vecteur

J'ai choisi le N = 131072 => 131072 éléments

- N= NombreBloc* NombreThread

NombreBloc et NombreThread	Temps (Seconds)
NombreBloc = 16384 et NombreThread = 8	1.191s
NombreBloc = 8192 et NombreThread = 16	1.159s
NombreBloc = 4096 et NombreThread = 32	1.127s
NombreBloc = 2048 et NombreThread = 64	1.153s
NombreBloc = 1024 et NombreThread = 128	1.149s



Dans l'addition vectorielle, les différences de performances sont très faibles. Lorsque la taille du bloc du threads diminue, le temps d'exécution diminue.

Exercice 2 : Addition de Matrice

J'ai choisi le N= 2048 => 4194304 éléments

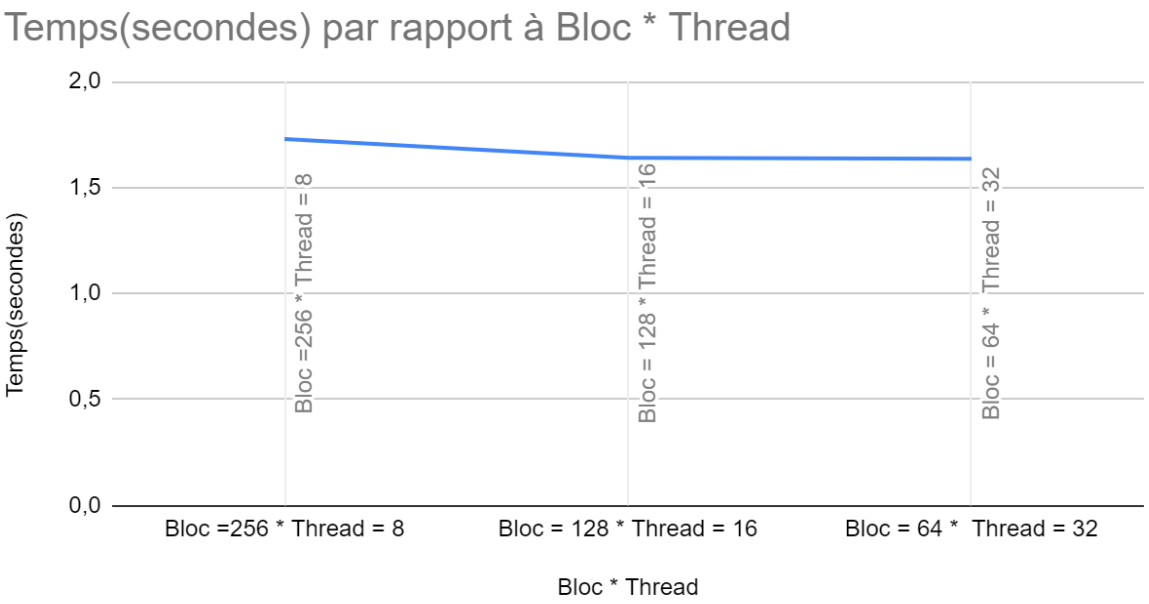
```
dim3 bD(NombreThread,NombreThread);
dim3 gD( NombreBloc,NombreBloc);
AdditionMatrice<<< gD ,bD>>>
```

- NombreBloc : c'est le nombre de blocs de threads dans une grille
- NombreThread : c'est le nombre de threads par block.

On peut remarquer que :

- Tant que le nombre de thread augmente et le nombre de blocs diminue le temps de calcul se réduit légèrement.
- Si on effectue notre calcul avec un nombre de thread supérieure à 1024(32*32) le calcul nous donne des zéro, donc ce noyau CUDA est limité à 1024 threads par bloc.

NombreBloc et NombreThread	**Temps (Seconds)**
NombreBloc = 256 et NombreThread = 8	1.728s
NombreBloc = 128 et NombreThread = 16	1.640s
NombreBloc = 64 et NombreThread = 32	1.636s
NombreBloc = 32 et NombreThread = 64	--



Exercice 3 : Multiplication Matrice GPU

J'ai choisi le N=1024 => 1048576 éléments

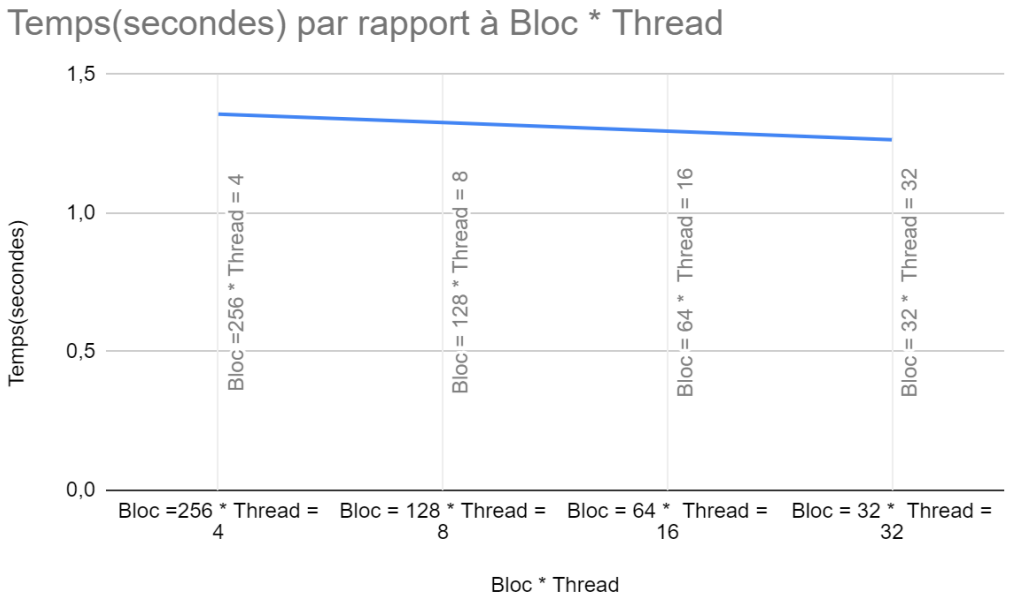
- **NombreBloc** : c'est le nombre de blocs de threads dans une grille
- **NombreThread** : c'est le nombre de threads par block.
- N= NombreBloc* NombreThread

```
dim3 bD(NombreThread,NombreThread);
dim3 gD( NombreBloc,NombreBloc);
AdditionMatrice<<< gD ,bD>>>
```

On peut remarquer :

- Quand le nombre de **thread augmente** et le **nombre de blocs diminue** le **temps de calcul se réduit**.
- Une différence de **0.997s**, entre le calcul avec **NombreThread=4** et celui avec **NombreThread=32**
- Si on effectue notre calcul avec un nombre de thread **supérieure à 1024(32*32)** le calcul nous donne des **zéro**, ce noyau **CUDA est limité à 1024 threads par bloc.**

NombreBloc et NombreThread	**Temps (Seconds)**
NombreBloc = 256 et NombreThread = 4	1.357s
NombreBloc = 128 et NombreThread = 8	1.327s
NombreBloc = 64 et NombreThread = 16	1.296s
NombreBloc = 32 et NombreThread = 32	1.265s
NombreBloc = 16 et NombreThread = 64	--



En ayant un grand nombre de blocs, ça oblige le streaming multiprocessors du GPU à passer beaucoup de temps à mettre en œuvre les blocs, mais en revanche, en augmentant le nombre de threads dans vos blocs plutôt que le nombre de blocs cela optimiser le temps de calcul, parce le surcoût de création de thread dans le GPU est minimale, car ils sont très légers.

- **Comparer avec la multiplication parallèle par bloc dans un CPU.**
  N=1024

NombreBloc et NombreThread	**Temps (Seconds)**
NombreBloc = 256 et NombreThread = 4	9.845s
NombreBloc = 128 et NombreThread = 8	11.932s
NombreBloc = 64 et NombreThread = 16	11.490s
NombreBloc = 32 et NombreThread = 32	13.131s
NombreBloc = 16 et NombreThread = 64	14.452s

Lorsque le nombre de bloc est réduit c'est-à-dire la taille des blocs est plus grande, le traitement complet prend plus de temps, car les données peuvent prendre beaucoup de temps à être traiter, et le coût de création de thread dans le CPU est élevé.