Optimiser les transferts de données

Optimisation de la mémoire utilisée (pinned memory), l'overlapping (copie asynchrone), zero-copy, Unified Virtual Addressing

Pinned memory

Pageable memory : mémoire dans l'hôte, qui est susceptible d'être copiée sur le disque dur (mémoire virtuelle) lorsque la RAM est saturée. Permet d'utiliser plus de mémoire qu'il n'est possible qu'avec la RAM.

Pinned memory : mémoire sur l'hôte qui ne peut pas être copiée sur le disque dur.

Demande d'accès GPU : par DMA (direct memory access), page-lock de la mémoire (interdit la copie), copie temporaire dans mémoire pinned (pinned host buffer). Pour éviter le problème, utilisant directement la pinned memory (taille plus importante que le pinned buffer) avec cudaHostAlloc (réservation et allocation) ou cudaHostRegister (mémoire déjà allouée).

Vérifier le type de mémoire utilisée : dans Command Line Profiler avec l'option memtransferhostmemtype. 0 = pageable memory, 1 = pinned memory.

Peu de pinned memory disponible, à utiliser avec précaution (peut ralentir le système complet). Quelle est la quantité de pinned memory disponible ?

Regouper les petits transferts de données

Surcoût fixe de transfert de données (latence). Regroupement des blocks de données permet de limiter cet overhead. Tester plusieurs taille de transfert.

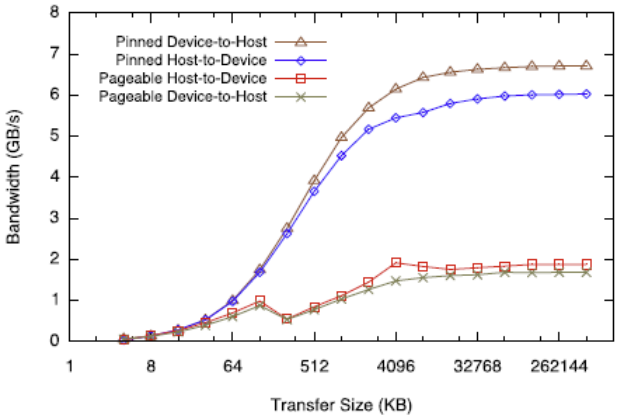


FIGURE 3.2

Host-to-device and device-to-host bandwidth for pageable and pinned memory versus transfer size on an Intel Xeon E5440 system with a Tesla K20.

Bande passante en fonction de la taille des blocs mémoire

Transferts asynchrones

Copie mémoire normalement bloquant. Possibilité d'utiliser non bloquant

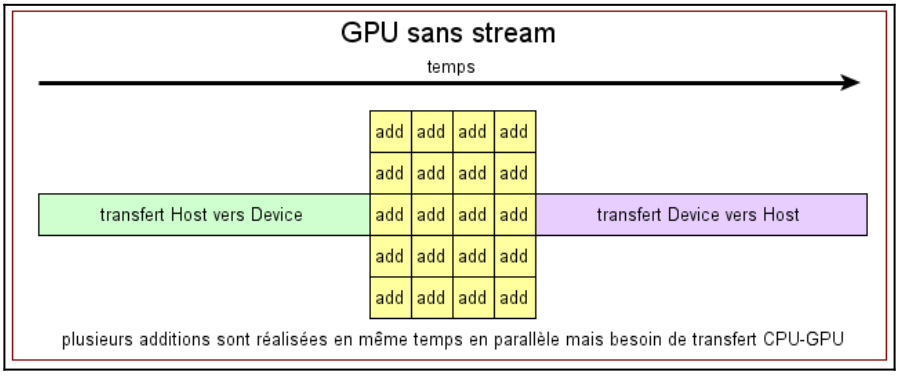
- cudaMemcpy() : synchrone, utilise pageable ou pinned memories
- cudaMemcpyAsync() : asynchrone, utilise pinned memory et streams

interleave : croise les transferts de données. overlapping : masque les transferts mémoires (réalisés en même temps que les calculs). Overlapping transfert et calculs possible sur toutes les versions de CUDA. Version récentes de CUDA : overlapping transferts de données in et out et calculs. Sur streams différents

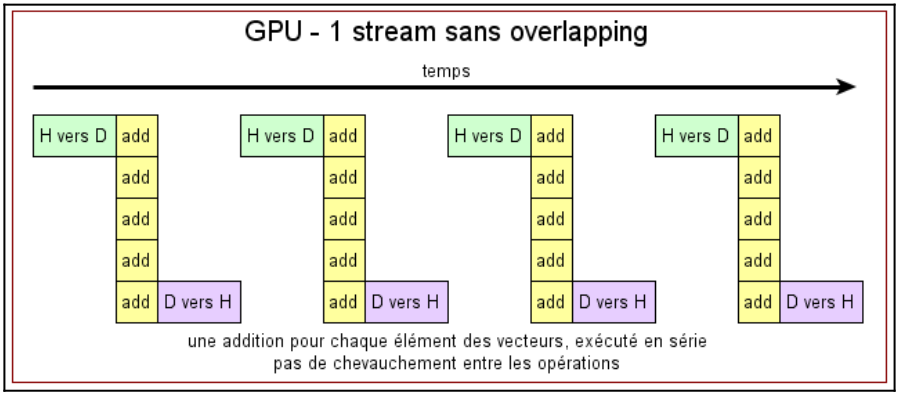
Comment tester si overlapping possible ? (et quel type d'ovelapping ?) : cudaDeviceProp.deviceOverlap.

Remarque : stream par défaut (0) est bloquant pour tous les streams (il attend que tous les autres streams ont fini avant de travailler)

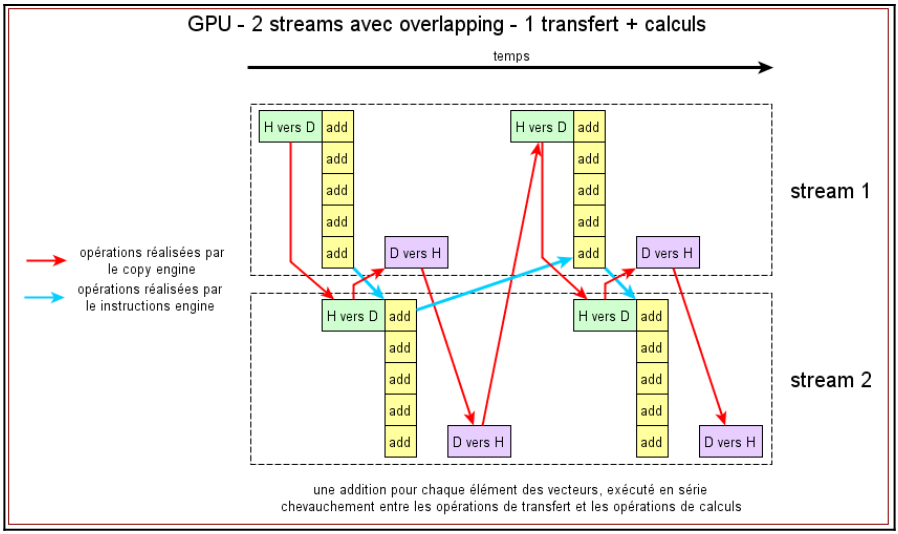
Voir avec le profiler comment fonctionne ces streams et l'overlapping :

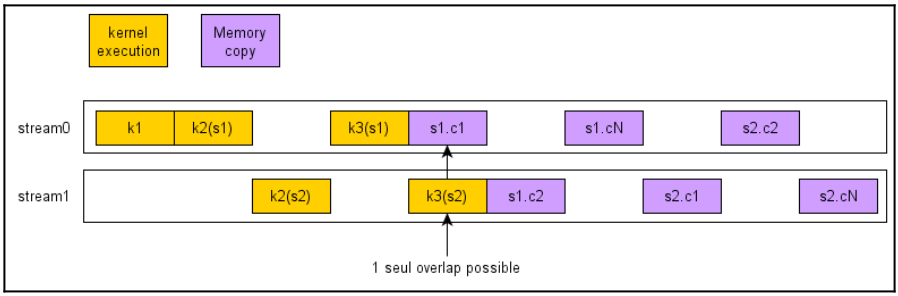


Transfert sans overlapping

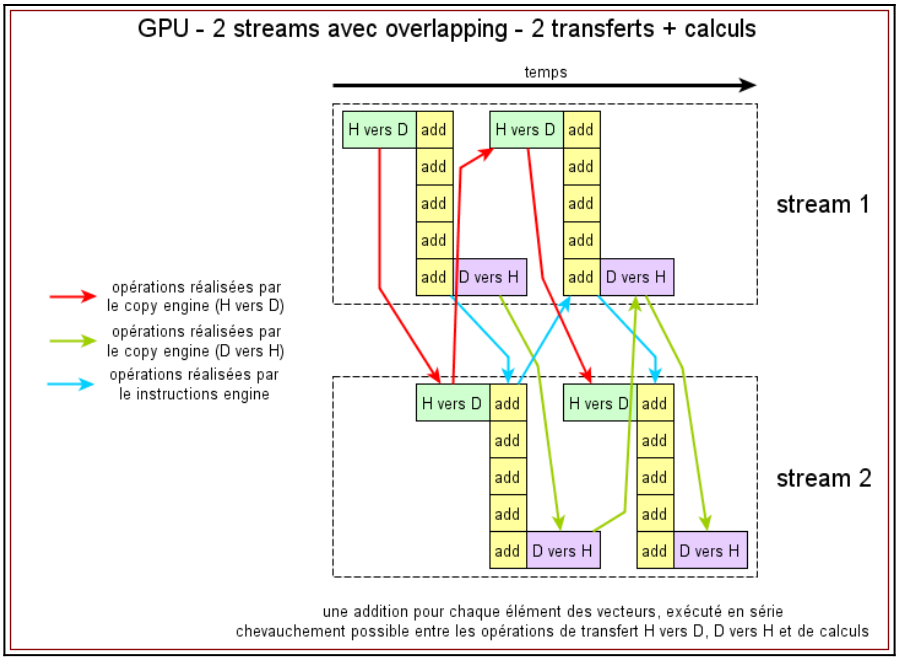


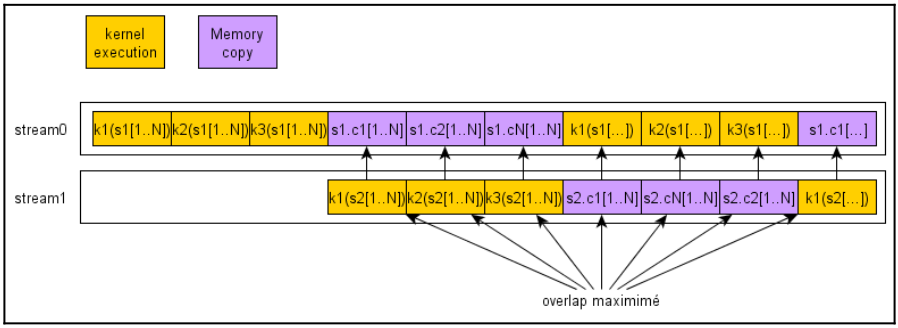
Transfert avec interleave





Overlapping simple (mémoires et caluls)





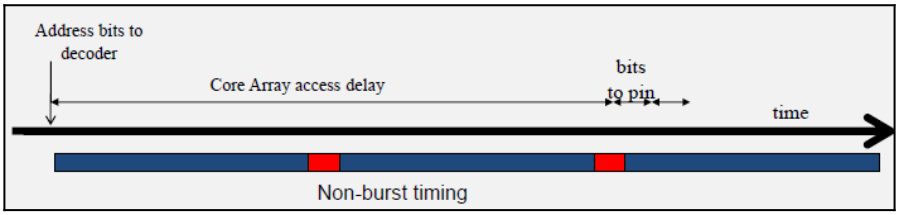
Overlapping double

Combien de streams utilisables ?

Lancement des instructions : lancer tous les commandes par streams (copy host-device, calculs, copy device-host) ou lancer par commandes (copy host-device sur tous les streams, puis tous les calculs sur tous les streams, puis tous les copy device-host sur tous les streams).

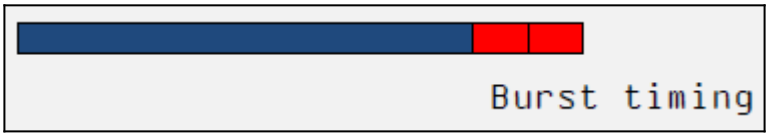
Burst memory

Accès aux banks de la mémoire globale : par paquet de 128 bits (à vérifier) = transaction. Sans burst :



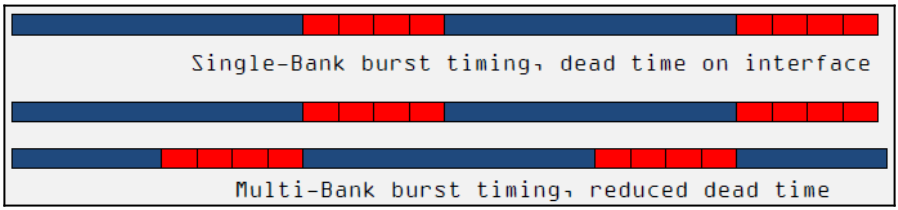
Sans burst

Avec burst, si données coalescente :



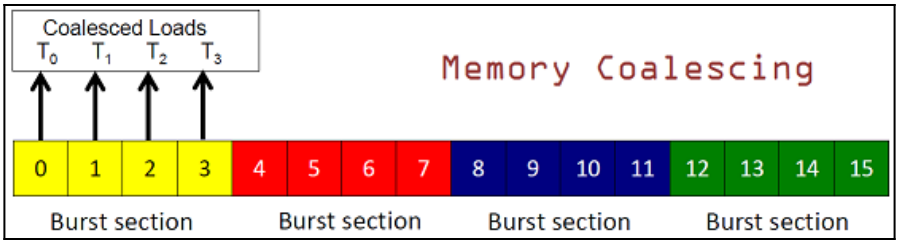
Burst simple

Utilisation de plusieurs banks pour remplir entre les burst :



Multi-bank burst

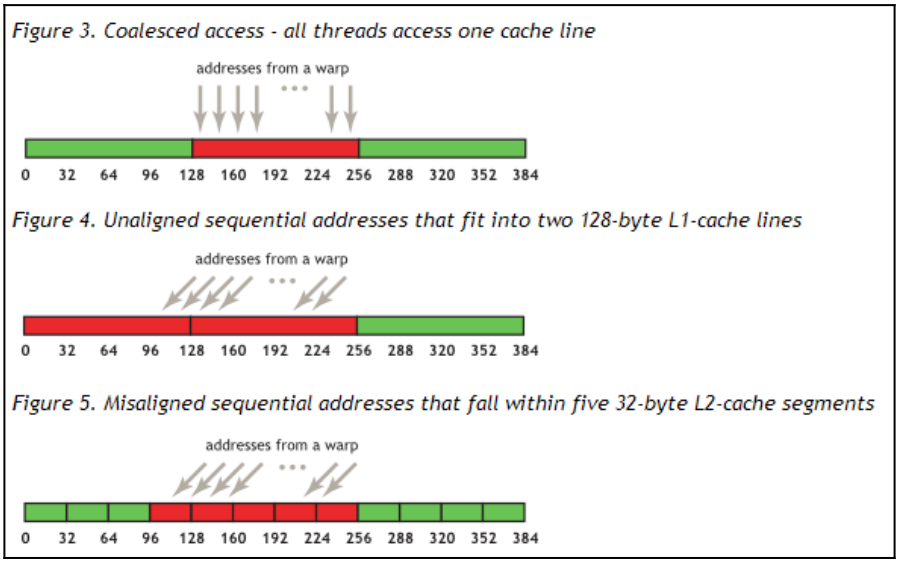
Avec les multi-bank :



Multi-bank et memory coalescence

Problèmes : non alignement des données sur les bursts. Nécessite plusieurs requêtes DRAM au lieu de 1. Mesure efficacité : numbre en pratique / nb théorique de burst. Coalscence si :

```
A[(terms independent of threadIdx.x)+ threadIdx.x];
```



Alignement

Par exemple, transformation 2D → 1D. Accès dans A non coalescent, accès dans B coalescent. Utilisation de sharde memory.

image ???

Conversion matrice 2D en mémoire 1D