

CentraleSupélec  

Mineure CalHau2

## CUDA : Fast transfers & overlapping

Stéphane Vialle

 universit  Paris-Saclay  Sciences et technologies de l'information et de la communication (STIC)  

Stephane.Vialle@centralesupelec.fr  
<http://www.metz.supelec.fr/~vialle>

1

---

---

---

---

---

---

---

---

---

---

CentraleSupélec

## CUDA :

### Fast transfers & overlapping

- 1 – Verrouillage de pages m moires CPU
- 2 – Transfert asynchrone CPU/GPU
- 3 – *Streams* & recouvrements

2

---

---

---

---

---

---

---

---


---

---

CentraleSupélec


Transfert rapide entre m moires CPU et GPU

### 1 – Allocation de m moire CPU *pinned*



- M moire pagin e standard (`malloc(...)`)
- M moire « *pinned* » (ou « *locked* ») → **transferts + rapides**
- M moire « *mapped* » dans l'espace CUDA du GPU
- M moire « *write-combined* »

La m moire pagin e peut  tre envoy e sur disque, alors que la m moire verrouill e (« *pinned* ») reste en RAM

 → Il ne faut pas verrouiller trop de gros volumes de donn es !  
 (sinon il n'y a plus de RAM disponible)

3

---

---

---

---

---

---

---

---

---

---

CentralesSupélec

Transfert rapide entre mémoires CPU et GPU

# 1 – Allocation de mémoire CPU *pinned*

**Le transfert CPU/GPU nécessite des pages mémoires verrouillées**

**Pageable Data Transfer**

**Pinned Data Transfer**

**Allocation de mémoire verrouillée et recopie page par page : op longue avec synchronisations**

Pour les données CPU à transférer :  
→ allouer dès le début de l'espace mémoire verrouillé

4

---

---

---

---

---

---

---

---

---

---

CentralesSupélec

Transfert rapide entre mémoires CPU et GPU

# 1 – Allocation de mémoire CPU *pinned*

**Allocation de mémoire « pinned » (ou « locked »)**

Allocation de mémoire verrouillée sur le CPU pour une application :

```
cudaMallocHost(AdrPtHost, size) ... deprecated
```

ou bien

```
cudaHostAlloc(AdrPtHost, size, cudaHostAllocDefault)
```

Allocation de mémoire verrouillée accessible à plusieurs applications (plusieurs contextes CUDA) :

```
cudaHostAlloc(AdrPtHost, size, cudaHostAllocPortable)
```

Libération de mémoire allouée dans la RAM du CPU

```
cudaFreeHost(PtHost)
```

*Rmq : on peut allouer de la mémoire verrouillée avec les bibliothèques systèmes CPU...mais avec CUDA c'est très simple à faire !*

5

---

---

---

---

---

---

---

---

---

---

CentralesSupélec

Transfert rapide entre mémoires CPU et GPU

# 1 – Conversion paginée ↔ *pinned*

**Conversion dynamique de mémoire paginée en mémoire « pinned »**

- On transforme en mémoire « pinned » un zone mémoire standard :
 

```
cudaHostRegister(PtHost, size, cudaHostRegisterDefault)
```

 ou
 

```
cudaHostRegister(PtHost, size, cudaHostRegisterPortable)
```
- On retransforme en mémoire paginée une zone « pinned » dès qu'on a plus besoin de la transférer vers/depuis le GPU:
 

```
cudaHostUnregister(PtHost)
```

 Evite de garder trop d'espace mémoire verrouillé et de manquer de RAM !

→ **Facilite la récupération/transformation de legacy codes !**

Inutile de modifier un code CPU récupéré et complexe

6

---

---

---

---

---

---

---

---

---

---

CentraleSupélec

## CUDA :

### Fast transfers & overlapping

- 1 – Verrouillage de pages mémoires CPU
- 2 – **Transfert asynchrone CPU/GPU**
- 3 – *Streams* & recouvrements

7

---

---

---

---

---

---

---

---

CentraleSupélec

Transfert rapide entre mémoires CPU et GPU

## 1 – Exploitation de mémoire CPU *pinned*

**3 exploitations possibles d'une zone mémoire CPU « pinned » :**

- 1 – Continuer à réaliser des transferts synchrones ... mais plus rapides  
`cudaMemcpy(PtDst, PtSrc, size, cudaMemcpyHostToDevice)`  
`cudaMemcpy(PtDst, PtSrc, size, cudaMemcpyDeviceToHost)`
- 2 – Réaliser des transferts asynchrones (non bloquants) sur le « *stream par défaut* », et réaliser une autre tâche en parallèle sur le CPU  
`cudaMemcpyAsync(PtDst, PtSrc, size, cudaMemcpy..To..)`  
ou  
`cudaMemcpyAsync(PtDst, PtSrc, size, cudaMemcpy..To.., /* stream = */ 0)`  
`kernel<<<Dg,Db>>>(...)`  
ou  
`kernel<<<Dg,Db,0, /* stream = */ 0>>>(...)`

**Sérialisation et synchro implicite des op lancées sur un même *stream***  
→ il y aura bien transfert asynchrone PUIS lancement d'un kernel

8

---

---

---

---

---

---

---

---

CentraleSupélec

Transfert rapide entre mémoires CPU et GPU

## 1 – Exploitation de mémoire CPU *pinned*

**3 exploitations possibles d'une zone mémoire CPU « pinned » :**

- 3 – Utiliser de «  *multiples streams* » pour exploiter au maximum le recouvrement des transferts et des calculs sur GPU

On peut créer puis utiliser jusqu'à 16 *streams*

```

cudaStream_t Stream[Ns];
cudaStreamCreate(&Stream[i])
cudaMemcpyAsync(PtGPU, PtCPU, size, cudaMemcpyHostToDevice, Stream[i]);
kernel<<<Dg,Db,0, Stream[i]>>>(...);
cudaMemcpyAsync(PtCPU, PtGPU, size, cudaMemcpyHostToDevice, Stream[i]);
cudaStreamDestroy(Stream[i]);

```

↑  
séríalísation

9

---

---

---

---

---

---

---

---

CentraleSupélec

## CUDA :

### Fast transfers & overlappings

- 1 – Verrouillage de pages mémoires CPU
- 2 – Transfert asynchrones CPU/GPU
- 3 – **Streams & recouvrements**

10

---

---

---

---

---

---

---

---

CentraleSupélec

Transfert rapide entre mémoires CPU et GPU

### 2 – Streaming

```

cudaMemcpy(a_d, a_h, N*sizeof(float), dir);
kernel<<< N/nThreads, nThreads >>>(a_d);

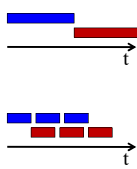
```

```

size = N*sizeof(float)/nStreams;
for (i = 0; i < nStreams; i++) {
    offset = i * N/nStreams;
    cudaMemcpyAsync(a_d + offset,
                    a_h + offset,
                    size, dir,
                    stream[i]);
}
for (i = 0; i < nStreams; i++) {
    offset = i * N/nStreams;
    kernel<<< {N/nStreams/BlkSizeX,1,1},
              {BlkSizeX,1,1}, 0,
              stream[i] >>>(a_d+offset);
}

```

**CUDA C Best Practices Guide**



Les ops. sur un même *stream* sont sérialisées automatiquement

16 *streams* sont utilisables

- **cudaStreamCreate**(...)
- **cudaStreamDestroy**(...)
- **cudaStreamSynchronize**(...)
- ...

11

---

---

---

---

---

---

---

---

CentraleSupélec

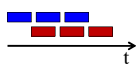
Transfert rapide entre mémoires CPU et GPU

### 2 – Streaming

```

size = N*sizeof(float)/nStreams;
for (i = 0; i < nStreams; i++) {
    offset = i * N/nStreams;
    cudaMemcpyAsync(..., stream[i]);
}
for (i = 0; i < nStreams; i++) {
    offset = i * N/nStreams;
    kernel<<< ..., stream[i]>>>(...);
}

```



Les ops. sur un même *stream* sont sérialisées automatiquement

Mais pas les ops sur d'autres *stream* :

→ 1<sup>ère</sup> sol : **synchro** explicite sur chaque *stream*

```

// Wait for all ops on all streams are
// completed
for (i = 0; i < nStreams; i++)
    cudaStreamSynchronize(stream[i]);
// Get global results on CPU
cudaMemcpy(CPU_res, GPU_res, size,
            cudaMemcpyDeviceToHost);

```

12

---

---

---

---

---

---

---

---

CentraleSupélec

Transfert rapide entre mémoires CPU et GPU

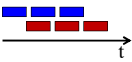
## 2 – Streaming

```

size = N*sizeof(float)/nStreams;
for (i = 0; i < nStreams; i++) {
    offset = i * N/nStreams;
    cudaMemcpyAsync(..., stream[i]);
}
for (i = 0; i < nStreams; i++) {
    offset = i * N/nStreams;
    kernel<<< ..., stream[i]>>>(...);
}

// Wait for all GPU ops are completed
cudaDeviceSynchronize();
// Get global results on CPU
cudaMemcpy(CPU_res, GPU_res, size,
           cudaMemcpyDeviceToHost);

```



Les ops. sur un même *stream* sont sérialisées automatiquement

Mais pas les ops sur d'autres *stream*:  
 → 2<sup>ème</sup> sol : *synchro* explicite avec tout le GPU

13

---

---

---

---

---

---

---

---

CentraleSupélec

## CUDA : Fast transfers & overlapping

# Fin

14

---

---

---

---

---

---

---

---