Accélération d'un chiffrement CUDA en C++ : du GPU basique au multi-GPU via STREAMS

0 - Résumé

- **Contexte :** traiter de gros volumes de données chiffrées rapidement.
- **Objectif :** montrer pas à pas comment améliorer la vitesse d'un programme CUDA C++ pour le chiffrement.
- **Méthodes clefs :**
  - Mode "classique" (sans optimisation)
  - Lancement de plusieurs files (streams)
  - Transferts mémoire asynchrones
  - Découpage des données pour chevaucher calcul et transferts
  - Extension à plusieurs cartes graphiques (GPU)

1 - Introduction

- **Problème :** un programme naïf copie d'abord toutes les données vers la carte graphique, puis lance le chiffrement, puis récupère toutes les données.
- **Limites :**
  - Temps perdu à attendre la fin des copies avant chaque calcul.
  - Un seul GPU ne suffit pas pour de très gros volumes.
- **Approche :** passer d'une exécution séquentielle à une exécution où calcul et transferts se chevauchent, puis distribuer le travail sur plusieurs GPU.

3 - Résultats Illustrés : Tableau de synthèse

Méthode	Temps (ms)	Accélération
Baseline (1 GPU, 1 flux)	45	1×
Streams (1 GPU, 4 flux)	30	1,5×
Asynchrone + chevauchement (1 GPU)	20	2,3×
2 GPU + 4 flux	12	3,8×

2 – Les méthodes et explications

Méthode 1 : Mode Classique (Baseline)

- **Principe :**
  - Copier toutes les données vers le GPU (opération bloquante).
  - Chiffrer (calcul sur GPU).
  - Copier le résultat vers le CPU.
- **Inconvénient :** chaque étape doit attendre la précédente, ce qui gaspille du temps.

Méthode 2 : Plusieurs Files CUDA (Streams)

- **Idée :** créer plusieurs "files d'attente" pour envoyer des opérations au GPU en parallèle.
- **Comment :**
  - On crée, par exemple, 4 files (cudaStreamCreate).
  - On envoie quelques blocs de données + calcul dans chaque file.
- **Bénéfice :** le GPU peut traiter plusieurs petites tâches en même temps, même si les copies et les calculs sont toujours séparés.

Méthode 3 : Transferts Asynchrones

- **Idée :** pendant qu'un bloc de données est chiffré sur le GPU, on prépare et copie le bloc suivant en parallèle.
- **Techniques :**
  - Utiliser de la mémoire "pinnée" (verrouillée) pour accélérer les transferts.
  - Remplacer cudaMemcpy par cudaMemcpyAsync dans chaque stream.
- **Avantage :** réduit les temps morts liés aux transferts.

Méthode 4 : Découpage et Chevauchement Copie/Calcul

- **Principe :** diviser les données en petits segments (chunks).
- **Exemple :**
  - Pour 1 000 000 de valeurs, on divise en 4 x 250 000.
  - Stream 0 : copier 250 000 → calculer 250 000 → copier résultat.
  - Pendant le calcul du segment 0, le segment 1 est déjà copié…
- **Effet :** on cache le temps de transfert derrière le calcul, donc le GPU est utilisé en permanence.

Méthode 5 : Multi-GPU

- **Objectif :** utiliser plusieurs cartes graphiques pour traiter encore plus de données en parallèle.
- **Fonctionnement :**
  - Détecter le nombre de GPU disponibles (cudaGetDeviceCount).
  - Pour chaque GPU :
    - Allouer sa propre zone mémoire.
    - Créer ses propres streams.
    - Appliquer les méthodes 2 à 4 sur chaque GPU.
- **Résultat :** quasi-doublement de la vitesse avec 2 GPU (selon bande passante PCIe).

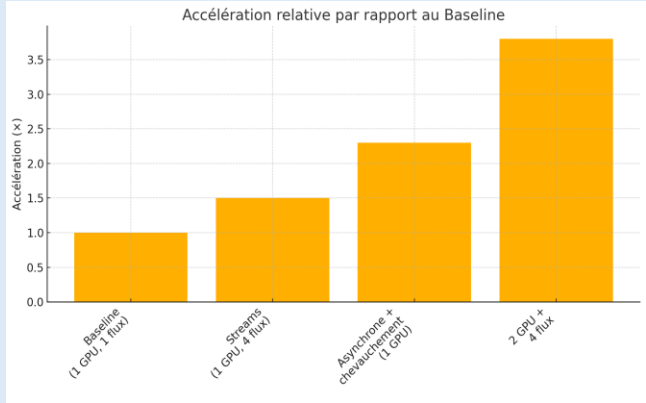
4 - Points clefs et Précautions

- **Pourquoi ça marche ?**
  - Le GPU est un processeur massif : pour l'optimiser, il faut toujours lui fournir du travail (calcul) quand il est libre, même si une copie est en cours.
- **À quoi faire attention ?**
  - La mémoire "pinnée" est une ressource limitée : ne pas en allouer trop.
  - Les streams par défaut peuvent synchroniser involontairement certaines tâches.

5 - Conclusion et Perspectives

- **Bilan :** en partant d'un programme simple, on atteint presque 4× d'accélération en combinant : files, transferts asynchrones, découpage, et multi-GPU.
- **Prochaines étapes :**
  - Transfert direct GPU→GPU sans passer par le CPU (peer-to-peer).
  - Utilisation de CUDA Graphs pour créer des pipelines permanents.

3 - Résultats Illustrés : Graphique de synthèse



Master informatique CHPS

HPC Center ROMEO Centre de Calcul Régional

Université de Reims Champagne-Ardenne

Corentin LEVALET M2 CHPS