

GPGPU avec les algorithmes STL

Algorithmes parallèles STL avec C++ 20

Michaël El Kharroubi

27.02.2024

HPC 2024 - HEPIA

Introduction



Maintenant que nous avons les bases, nous allons voir comment paralléliser notre code avec le kit HPC de Nvidia

Le SDK HPC de Nvidia

À quoi sert ce kit?

Ce kit contient :

- des librairies
- des compilateurs (C, C++, Fortran)
- des débogueurs
- des outils de profiling
- des librairies de communication (ex : OpenMPI)

En résumé, un grand nombre d'outils pour réaliser une application HPC (basé sur du matériel Nvidia majoritairement)

À quoi sert ce kit?

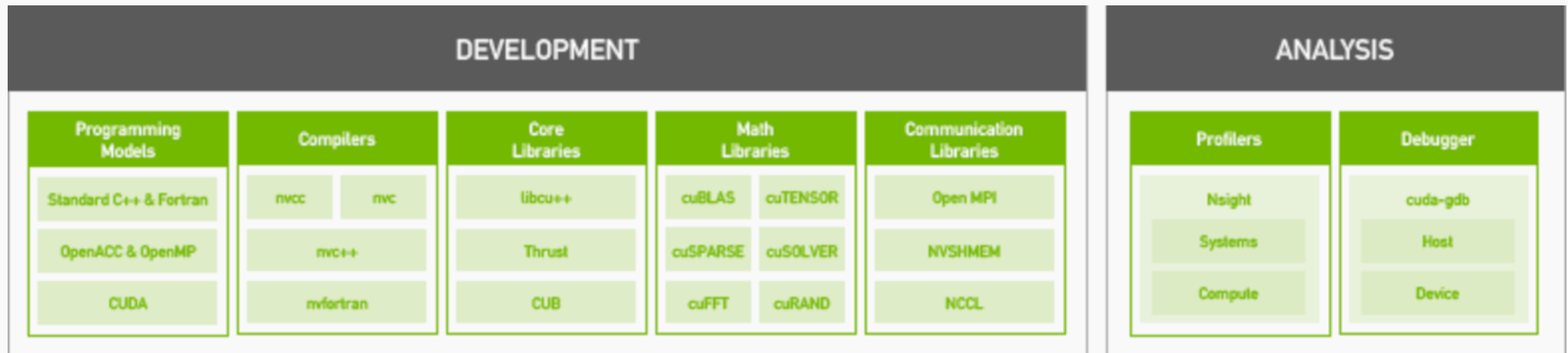


Figure 1: Contenu du HPC SDK de Nvidia - Source : Nvidia

Dans le cadre de ce cours, nous allons nous concentrer sur le compilateur nvc++

Il permet de compiler du C, C++ ou CUDA

Ce qui va nous intéresser particulièrement, c'est son support des *Parallel Algorithms* introduits avec C++17

Exemple de parallélisation avec nvc++

Les politiques d'exécution

Depuis C++17, les algorithmes de la STL peuvent prendre en paramètre une politique d'exécution. Cette politique permet d'indiquer au compilateur comment paralléliser le traitement des données

On retrouve 4 politiques :

| seq | par | par_unseq | unseq (C++ 20) |
|---|--|--|--|
| <ul style="list-style-type: none">• Exécution séquentielle simple | <ul style="list-style-type: none">• Exécution en parallèle | <ul style="list-style-type: none">• Exécution en parallèle avec possibilité d'utiliser des instructions vectorielles | <ul style="list-style-type: none">• Exécution séquentielle avec possibilité d'utiliser des instructions vectorielles |

Exemple d'algorithme STL avec une politique d'exécution

```
std::vector<int> v = {6, 3, 4, 12, 1, 15};

std::sort(std::execution::par,
          v.begin(), v.end());

std::vector<int> v_sorted = {1, 3, 4, 6, 12, 15};

bool is_sorted = std::equal(std::execution::par_unseq,
                             v.begin(), v.end(),
                             v_sorted.begin());

std::cout << is_sorted << std::endl;
```

Une fois notre code écrit, il faut le compiler avec nvc++ en indiquant comment le paralléliser

```
nvc++ -std=c++20 -stdpar=gpu -O3 parallel_sort.cpp -o parallel_sort
```

Nous avons trois paramètres principaux :

- `-std=c++20` : On utilise le dialecte C++20.
- `-stdpar=gpu` : On veut que le code soit parallélisé sur GPU
- `-O3` : On veut que le compilateur utilise le niveau 3 d'optimisation sur notre code

Les types de parallélisation disponibles

Le compilateur nvc++ permet de paralléliser son code sur CPU ou sur GPU avec le framework Thrust de Nvidia

Le paramètre `stdpar` peut prendre 2 valeurs :

multicore

Le code est parallélisé sur un CPU multicore

gpu

Le code est parallélisé sur GPU. Nécessite un GPU Nvidia avec une architecture Pascal (ex: GTX 1080) ou plus récent

Exemple d'analyse d'exécution sur GPU avec nvprof

Comment passer des données au GPU avec les captures

Communiquer des données au GPU

Pour rappel, le GPU et CPU ont deux espaces d'adressage distincts. On ne peut donc pas directement accéder à des données qui se trouve sur la ram de l'*Host* depuis le *Device*

Quand on passe en paramètre un itérateur à un algorithme STL, il n'y a rien besoin de faire. Le compilateur va gérer l'allocation, la copie et la libération des données sur le device

Mais comment faire pour capturer un vecteur par exemple ?

Exemple d'accès illégal

```
std::vector<int> v = {6, 3, 4, 12, 1, 15};  
std::vector<int> idxs(v.size(), 0);  
  
std::iota(idxs.begin(), idxs.end(), 0);  
  
int cst = 4;  
  
std::for_each(std::execution::par_unseq,  
              idxs.begin(), idxs.end(),  
              [&v, &cst](int i){  
                v[i] += cst;  
              });
```


Exemple d'accès illégal

Ce code provoque un warning à la compilation et une erreur `cudaErrorIllegalAddress` à l'exécution

Solution

Pour régler ce problème, il suffit de capturer la mémoire par copie

```
std::vector<int> v = {6, 3, 4, 12, 1, 15};  
std::vector<int> idxs(v.size(), 0);  
  
std::iota(idxs.begin(), idxs.end(), 0);  
  
int cst = 4;  
  
std::for_each(std::execution::par_unseq,  
              idxs.begin(), idxs.end(),  
              [device_v = v.data(), cst](int i) {  
                device_v[i] += cst;  
              });
```

Le compilateur ne peut pas tout copier

Attardons nous sur cet extrait de code “[`device_v = v.data()`, `cst`]”

Le compilateur ne va pas simplement copier le pointeur `v.data()`, il va remonter à l'allocation mémoire de la zone pointée et va s'occuper de la rendre accessible depuis le device

Par conséquent, il n'est pas possible d'utiliser de la mémoire allouée par du code qui n'aurait pas été compilé par `nvc++` (ex: une librairie partagée comme OpenCV)

Le compilateur ne peut pas tout copier

Le compilateur s'appuie principalement sur la CUDA Unified memory. Pour plus d'information à ce sujet, je vous recommande ce billet de blog : <https://developer.nvidia.com/blog/unified-memory-cuda-beginners>

Le futur avec C++23 (views et
mdspans)

La prochaine étape majeure dans ce paradigme apparaît avec C++23. Depuis C++ 20, les spans et les views ont fait leur apparition

Les views sont une forme d'itérateurs à évaluation paresseuse

Dans C++23, on voit apparaître de nouvelles méthodes pour combiner les views comme par exemple le produit cartésien pour générer les indices d'une matrice

Et la grande nouveauté, les mdspans qui permettent de gérer des structures de données multidimensionnelles

Exemple de code C++23 et notions à
retenir

Notions essentielles à retenir

- On parallélise notre code avec le compilateur `nvc++` du SDK HPC de Nvidia
 - Il faut passer le paramètre `stdpar` au compilateur pour lui dire comment paralléliser (`cpu`, `gpu`).
- Pour paralléliser un algorithme STL, il faut lui passer une politique d'exécution
- Connaître et comprendre la différence entre les 4 politiques d'exécution
- On ne peut pas capturer des données du *host* par référence, et les utiliser sur le *device*
- On capture les zones mémoire par copie pour les passer au *device*
- On ne peut pas capturer de la mémoire allouée par un code qui n'a pas été compilé par `nvc++`.

Questions ?
