UF I5AISE51
Dimensionnement et évaluation des architectures
Introduction à la programmation massivement parallèle

—

Partie 6
Algorithme parallèle de réduction

—

P.-E. Hladik

INSA Toulouse

11 janvier 2021

Plan

Qu'est-ce q'un algorithme de réduction ?

Agglomérer un ensemble de valeurs d'entrée en une seule valeur en utilisant une "opération de réduction", exemples :

- Max
- Min
- Somme
- Produit

Algorithme séquentiel en O(N)

1. Initialiser le résultat en tant que valeur d'identité pour l'opération de réduction :
   - La plus petite valeur possible pour le Max
   - Valeur la plus élevée possible pour le Min
   - 0 pour la Somme
   - 1 pour le Produit
2. Itérer sur les valeurs d'entrée l'opération de réduction entre la valeur du résultat et la valeur actuelle

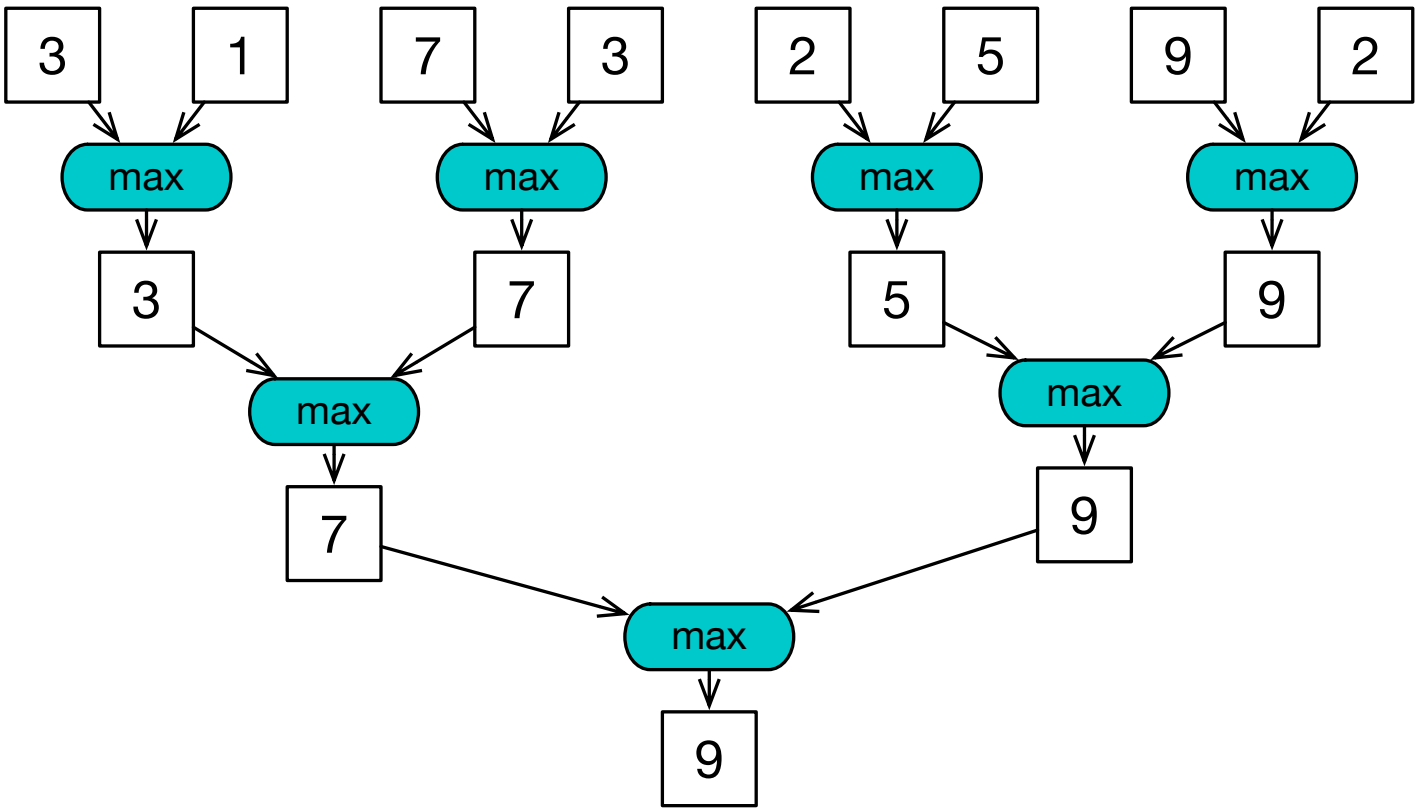
- N opérations de réduction effectuées pour N valeurs d'entrée
- Chaque valeur d'entrée n'est visitée qu'une seule fois – algorithme en O(N)
- Il s'agit d'un algorithme efficace sur le plan du calcul

Stratégie de partionnement et d'aglomération

Une stratégie couramment utilisée pour le traitement de grands ensembles de données d'entrée

- pas d'ordre requis des éléments de traitement dans un ensemble de données
- Diviser l'ensemble de données en plus petits morceaux
- Avoir un thread pour traiter un morceau
- Construire un arbre de réduction pour agloméré les résultats de chaque morceau

Algo parallèle d'arbre de réduction en log(N) étapes



Analyse rapide

Pour N valeurs l'arbre de réduction effectue :

- $(1/2)N + (1/4)N + (1/8)N + \ldots = (1\text{-}(1/N))N = N\text{-}1$ opérations en Log (N) étapes
  - Exemple : le traitement de 1 000 000 de valeurs prend 20 étapes (en supposant que nous disposions de ressources d'exécution suffisantes)
- Parallélisme moyen $(N\text{-}1)/\text{Log}(N))$
  - Pour N = 1 000 000, le parallélisme moyen est de 50 000
  - Toutefois, le pic de ressources nécessaires est de 500 000
- C'est un algorithme paralèle performant en terme de travail
  - La quantité de travail effectuée est comparable à celle d'un algorithme séquentiel efficace
  - De nombreux algorithmes parallèles ne fonctionnent pas efficacement

Exemple : réduction d'une somme

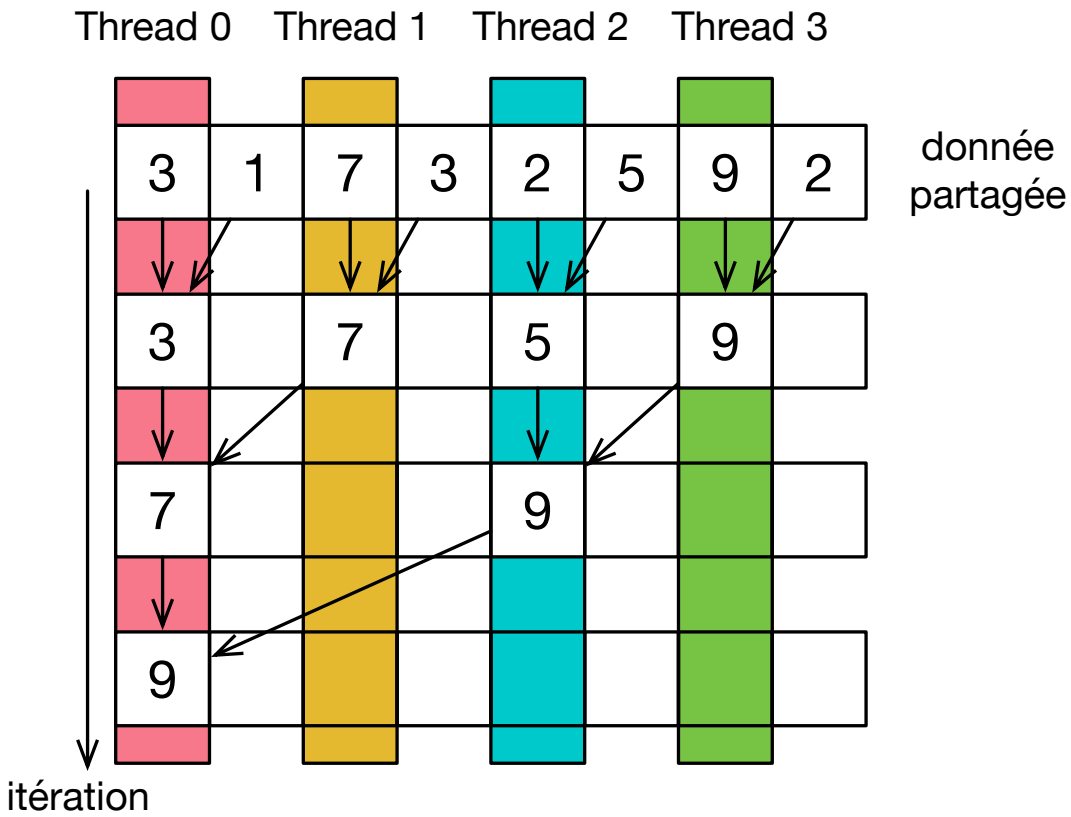
Implémentation parallèle

- Chaque thread additionne 2 valeurs à chaque étape
- Réduit par 2 le nombre de thread à chaque étape
- Utilise $\log(n)$ étapes pour n éléments et $n/2$ threads

Suppose une réduction en utilisant la mémoire partagée

- Le vecteur initial est dans la mémoire globale du device
- La mémoire partagée contient une somme partielle
- Au départ, la mémoire partagée contient le vecteur initial
- Chaque étape rapproche la somme partielle de la somme
- La somme finale sera dans l'élément 0 du vecteur partagée
- Réduit le trafic global de mémoire grâce à la lecture et à l'écriture de valeurs de somme partielle
- La taille maximale des blocs doit être inférieure ou égale à 2048

Exemple : réduction d'une somme (schéma)



Exemple : implémentation naïve

- Chaque block dispose de 2*BlockDim.x éléments
- Chaque thread charge 2 éléments dans la mémoire partagée

```
__global__
void redux(float* input, float* ouput) {
  __shared__ float partialSum[2*BLOCK_SIZE];

  int t = threadIdx.x;
  int start = 2*blockIdx.x*blockDim.x;

  partialSum[t] = input[start + t];
  partialSum[blockDim+t] = input[start + blockDim.x+t];

  for (int stride = 1; stride <= blockDim.x;  stride *= 2)
  {
    __syncthreads();
    if (t % stride == 0)
      partialSum[2*t]+= partialSum[2*t+stride];
  }
  ouput[blockIdx.x] = partialSum[0];
}
```

Commentaires sur l'implémentation naïve

A chaque itération, deux chemins de flux de contrôle seront parcourus séquentiellement pour chaque *warp* :

- Les threads qui effectuent des sommes et les threads qui n'en effectuent pas
- Les threads qui n'effectuent pas de somme consomment toujours des ressources d'exécution

La moitié ou moins des threads seront en cours d'exécution après la première étape

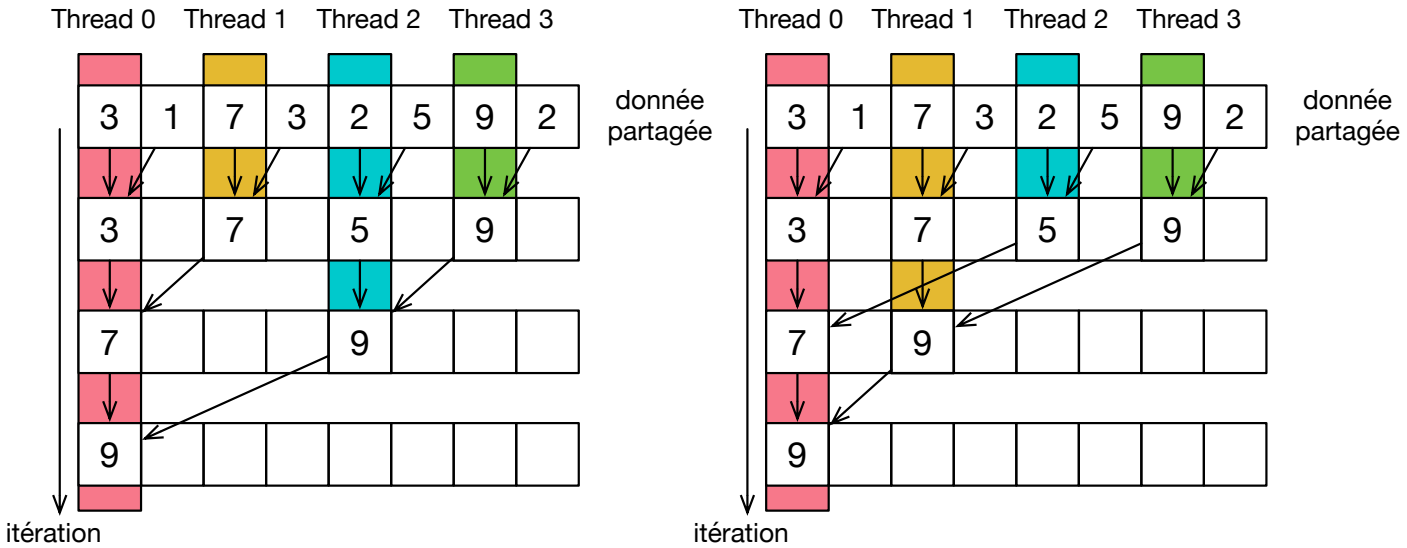
- Tous les threads d'indices impairs sont désactivés après la première étape
- Après la 5e étape, des warps entier dans chaque bloc échouent le test conditionnel (une mauvaise utilisation des ressources mais aucune divergence
- Cela peut durer un certain temps, jusqu'à 6 étapes supplémentaires (foulée = 32, 64, 128, 256, 512, 1024), où chaque warp active n'a qu'un thread "productif".

Warp

Warp

[wikipedia] On the hardware side, a thread block is composed of ‘warps’. A warp is a set of 32 threads within a thread block such that all the threads in a warp execute the same instruction. These threads are selected serially by the SM.

Toujours regrouper les threads "productifs"



Exemple : implémentation plus efficace

```
__global__
void redux(float* input, float* ouput) {
  __shared__ float partialSum[2*BLOCK_SIZE];

  int t = threadIdx.x;
  int start = 2*blockIdx.x*blockDim.x;

  partialSum[t] = input[start + t];
  partialSum[blockDim+t] = input[start + blockDim.x+t];

  for (int stride = blockDim.x; stride > 0; stride /= 2)
  {
    __syncthreads();
    if (t < stride)
      partialSum[t] += partialSum[t+stride];
  }
  ouput[blockIdx.x] = partialSum[0];
}
```