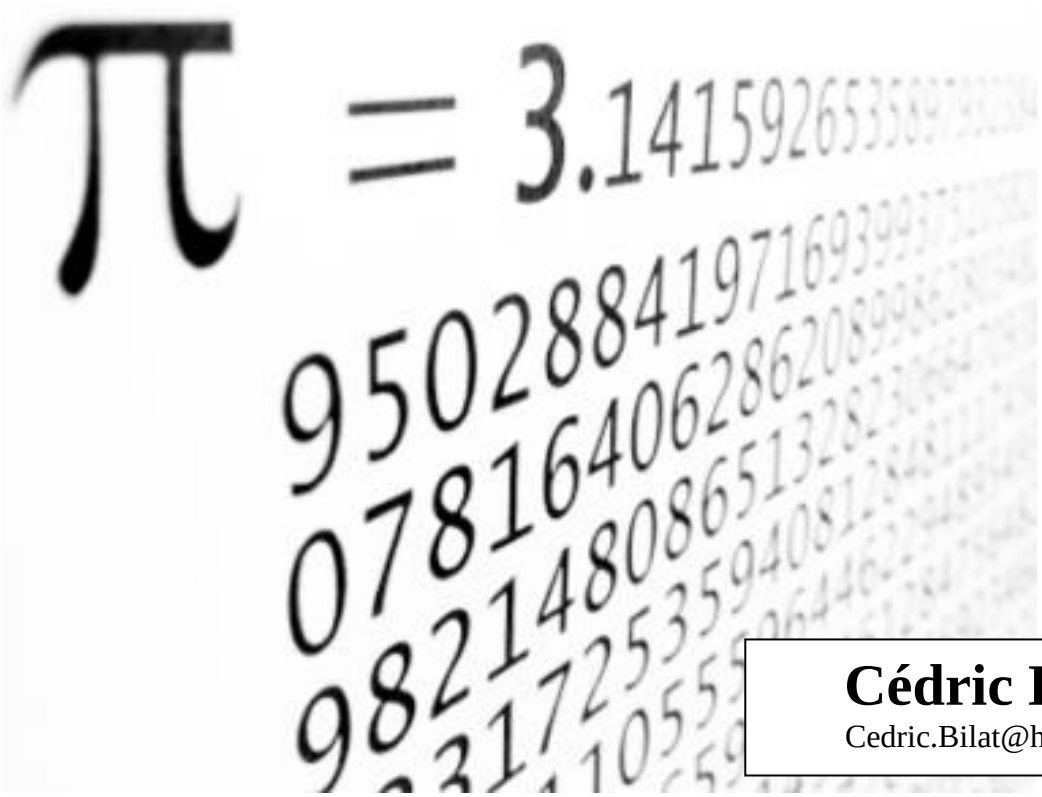
GPGPU



Cédric Bilat
Cedric.Bilat@he-arc.ch

Slice

GM

Prérequis

TP_Slice_Algo
TP_SliceGM_Host

But

Le but est d'effectuer une réduction parallèle côté *device* du tableau en GM

Solution

Technique

On effectue une réduction sur place par dichotomie et écrasement successif de *tabGM*. A La fin le résultat se trouvera dans la première case du tableau en GM.

Hypothèse

Cette approche requiert que la taille de *tabGM* soit une puissance de 2. Il faut donc que le nombre de thread que vous aurez choisi via la grille soit une puissance de 2, puisque avec la technique de promotion de tableau, chaque thread possède sa case dans *tabGM*.

Host Side

```
void SliceGM::run()
    {
    reductionIntraThread<<<dg,db>>>(nbSlice,tabGM); // assynchrone

    // ecrasement
          {
          int moitier=nTabGM/2;

         // une autre grid que ci-dessus
          dgx(moitier,1,1) ;      // 1D
          dbx(1,1,1) ;            // 1D , contrainte dbx.x <=1024

          while(moitier>=1)
              {
              ecrasementGM<<<dgx,dbx>>>(tabGM,moitier);//barriere synchronisation
              moitier=moitier/2;  // moitier=moitier>>1
              // TODO : udate du nombre de thread, dgx.x= ...
              // A vous de jouer !
              }
          }

    float result ;
    GM::memcpyDToH(&result, &tabGM[0], sizeof(float} ; // ou tabGM
    *ptrPiHat=result; // ou result*DX (selon votre code coté device)
    }
```

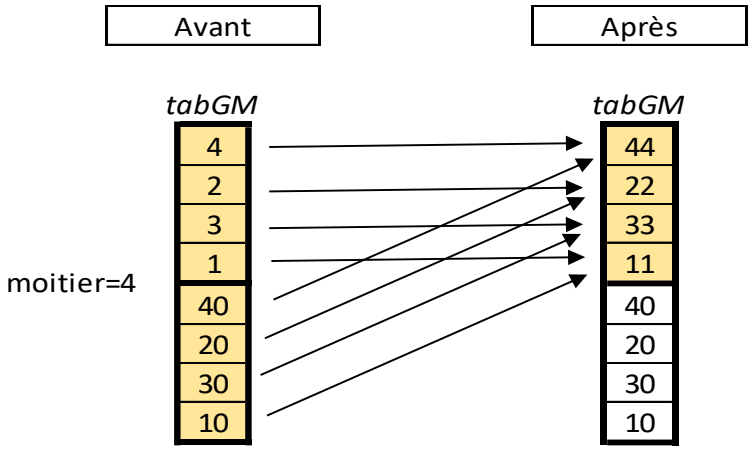
Device Side

A vous de produire le code coté device, ie à implémenter entre autre, le kernel écrasement !

```
__global__ void ecrasementGM(float* tabGM, int moitier);
```

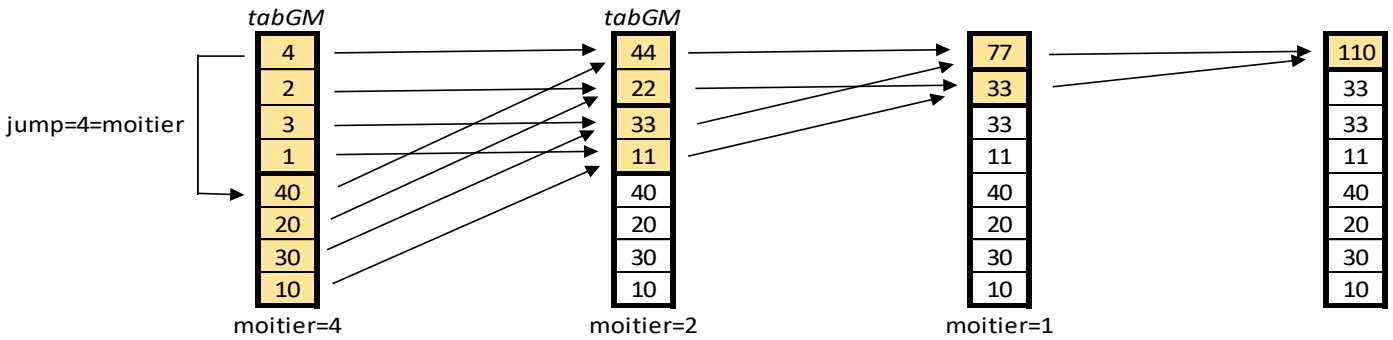
Indication

Cette kernel ramène toute la sémantique du tableau dans la première moitié du tableau.



La sémantique se trouve dans la partie jaune. La partie jaune de « après » n'est plus que dans la première moitié du tableau. La somme de la partie de jaune de « avant » est la même que la partie somme de « après ».

En recommençant cette même « réduction » sur *après*, la sémantique se trouvera que dans les deux premières cases, puis en recommençant encore, la sémantique ne se trouvera que dans la première case, et notre réduction de *tabGM* sera finie. Ces itérations sont assurées par le code côté host !



4+2+3+1+40+20+30+10=110

Note

(N1) Il s'agit d'une réduction parallèle de *tabGM* coté *device*. Plus on avance dans la réduction, moins de thread travaille. A la fin, il n'y a plus que 1 thread.

Piège

Ici on se trouve avec un code host utilisant deux kernels différents :

- reductionIntraThread
- ecrasement

On se retouve donc avec deux grilles différentes, une par kernel !

Squelettes de codes

Des squelettes de codes vous sont fournis dans le workspace.

1) Complétez les ***TODO*** (requiert de comprendre le code fournit et son organisation)
2) Faites passer les *uses*
3) Faites passer les *tests unitaires*

Concept

Les squelettes de codes vous sont fournis pour gagner du temps de saisie de code c++ standard. L'objectif dans ce cours est de mettre l'accent sur Cuda,

Il vous incombe néanmoins de comprendre les codes fournis, leur interaction et les structures sous-jacente.

Tests Unitaires

Les ***tests unitaires*** sont déjà codés.

Procédure

Dans le workspace, on peut spécifier deux types d'exécution :

- ***USE***
- ***TEST***

dans la méthode ***main*** à la racine du projet.

Etape 1

Passer le workspace en mode *test* dans ***main.cpp***, à la racine du projet.

Etape 2

Mettez en commentaire les tests qui ne vous intéressent pas dans ***mainTest.cpp***
Inutile de tous les lancer à chaque fois !
Idem pour **mainUse.cpp**

Etape 3

Choisissez une rendu *console* ou *html* dans ***mainTest.cpp***
(html conseillé)

Amélioration

Observation

(O1) Tous les calculs se font côté device (Bien !)
(O2) Lancer les écrasements côtés host nous reste un peu en travers de la gorge !
(O3) Impossible de respecter les heuristiques pour la grille utilisée pour la réduction intra-thread

Amélioration

Attendez le prochain TP !

End