Algorithmique pour GPU

Sylvain Collange
Journées ANR TaMaDi

27 octobre 2010

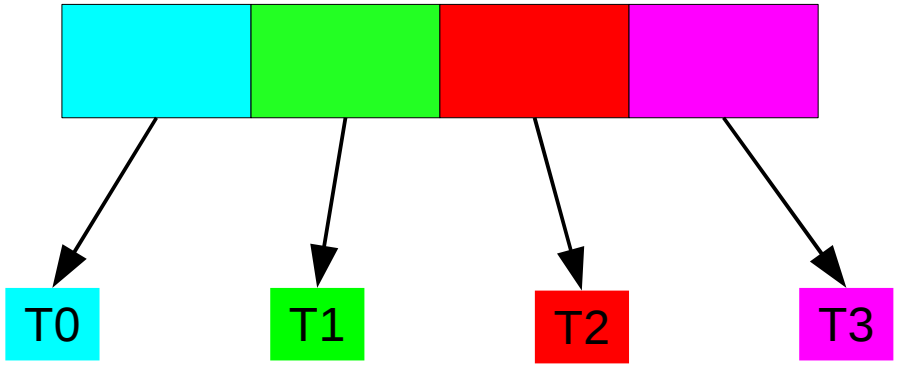
Arénaire, LIP

Introduction

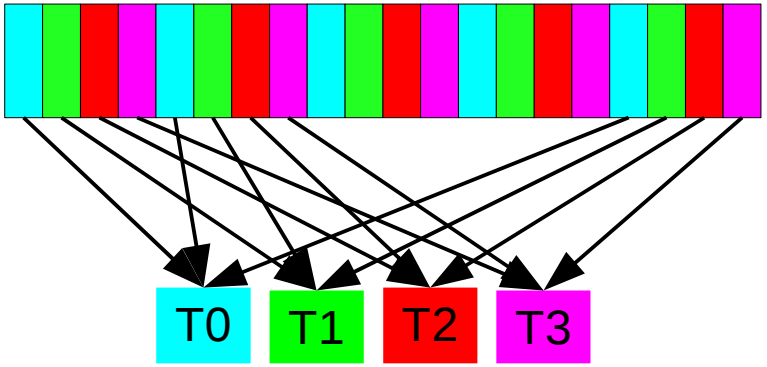
- Point de vue haut-niveau
  - Avant de commencer à programmer
  - Indépendant de l'architecture, langage
- Le calcul n'est pas cher, la mémoire l'est
  - Optimiser les mouvements de données en priorité
- Questions
  - Comment répartir le travail ?
  - Comment ranger les données ?
  - Combien de threads ?

Répartir le travail

- Sur CPU multicœur / multiprocesseur
  - Parallélisme à gros grain
  - **Découpler** les tâches pour limiter les **conflits** et communications



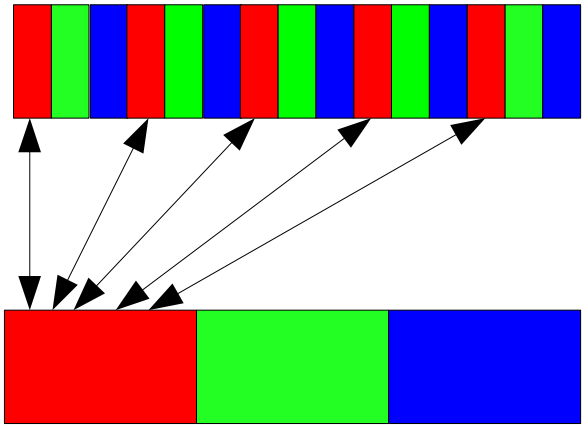
- Sur GPU
  - Parallélisme à grain fin
  - **Entrelacer** les tâches
  - Faire apparaître de la **localité** : exploiter les mémoires locales
  - Faire apparaître de la **régularité** : exploiter les unités SIMT



Ranger les données

- Tableau de structures (AoS)
  - Alignement ?
  - Accès partiel (bleu uniquement) ?
  - Motif d'accès sur GPU ?
- Structure de tableaux (SoA)
- NVIDIA : registres sous forme SoA
  - Conversions coûteuses
    - AoS → SoA = Gather
    - SoA → AoS = Scatter
    - Ou transposition en mémoire partagée
- Préférer SoA en mémoire

```
struct Pixel {
    float r, v, b;
};
Pixel image[480][640];
```



```
struct Image {
    float R[480][640];
    float V[480][640];
    float B[480][640];
};
Image image;
```

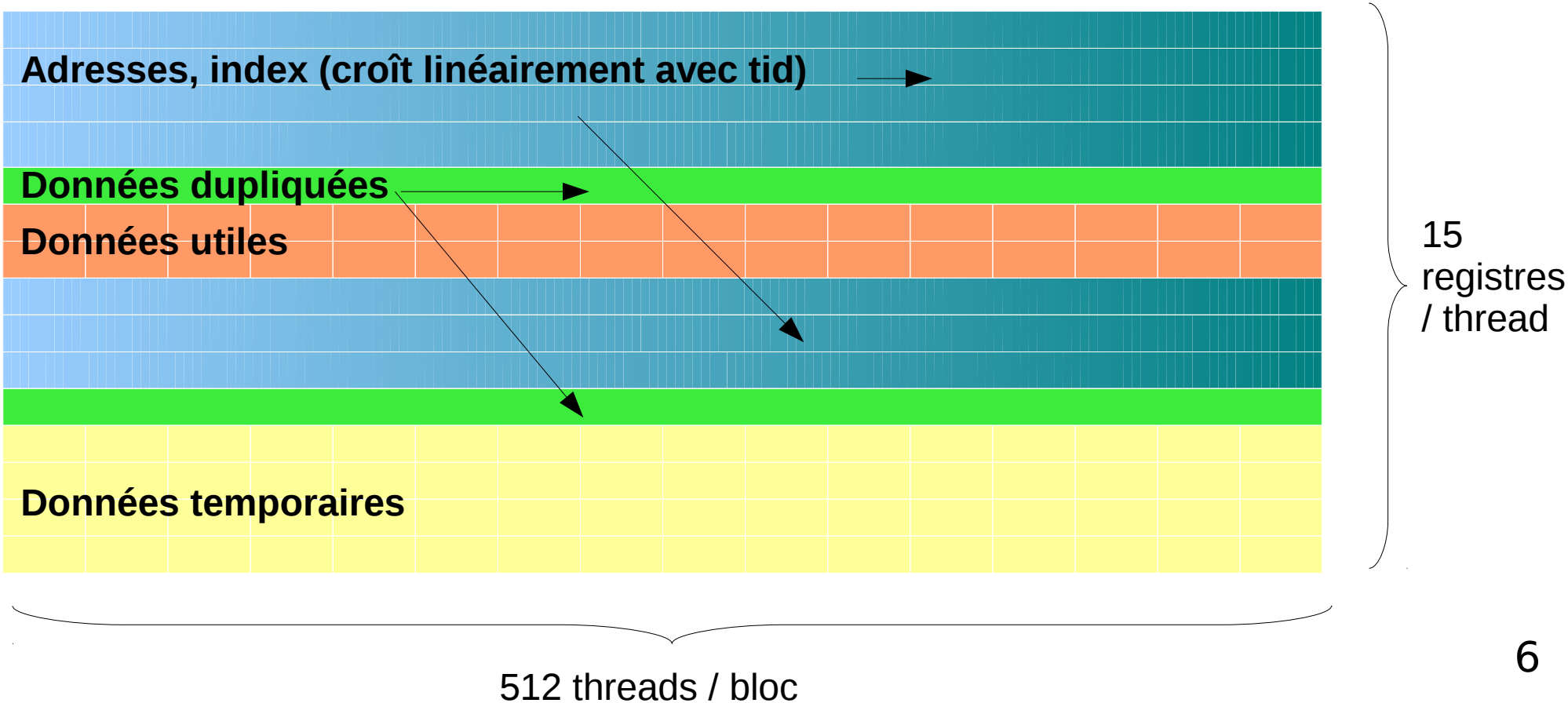
Combien de threads?

- Le plus possible ?
  - Parallélisme de données
    - Masque les latences
  - Localité
    - Stocker les données privées de chaque thread
  - Surcoût de gestion
    - Initialisation, calculs d'adresses, contrôle
    - Opérations redondantes
- Parallélisme d'instructions
  - Permet également de masquer les latences
    - Jusqu'à 5 lectures mémoire/thread en vol sur Tesla
    - Exécution superscalaire (supervectorielle ?)
  - Plus limité que sur CPU, mais présent !

Exemple : SGEMM de CUBLAS 1.1

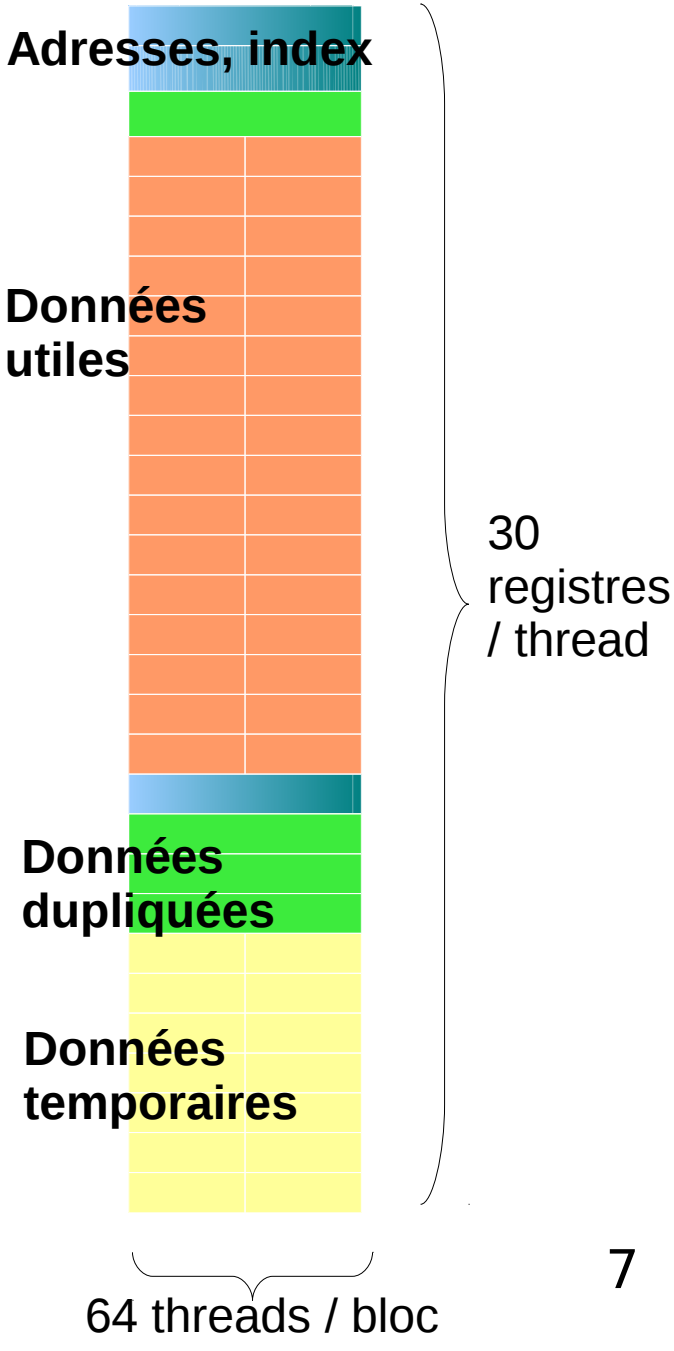
Tiré de: **Vasily Volkov**. Programming inverse memory hierarchy : case of stencils on GPUs. *ParCFD*, 2010.

- 512 threads / bloc, 15 registres / thread
- 9 registres / 15 contiennent des données redondantes
- Seuls 2 registres réellement utiles



Moins de threads, plus de calculs

- SGEMM Volkov
  - 8 éléments calculés / thread
  - Déroulage de boucle
  - Moins d'accès à mémoire partagée, plus dans registres
- Surcoût amorti
  - 1920 registres contre 7680 pour faire le même travail
  - Idem pour les opérations redondantes
- Success story
  - Perf +60% par rapport à CUBLAS 1.1
  - Intégré dans CUBLAS 2.0



Morale

- Bien distribuer son travail
  - Localité
  - Régularité
- Bien placer ses données
  - Privilégier SoA
  - Localité, régularité toujours
  - Règles de bon sens (éviter transferts et indirection inutiles...)
- Plus de threads ≠ plus de perf.
  - Saturer parallélisme d'instructions d'abord (quasi-gratuit)
  - Compléter par parallélisme de données (coûteux en mémoire)