Projet II : Batch merge path sort

Thomas Jacquot
Simon Demouchy

ENSAE Paris

21 Mai 2021

Introduction

- Implémentation en CUDA de l'algorithme présenté dans l'article :

  O.Green et al. "GPU Merge Path - A GPU Merging Algorithm"(2012)[1]

- Comparaison du temps d'exécution de cet algorithme "Merge Path" parallélisé avec l'équivalent en séquentiel.

- Problématique : *L'approche par parallélisation est-elle plus efficace que l'approche séquentielle ?*

Illustration du principe de Merge Path

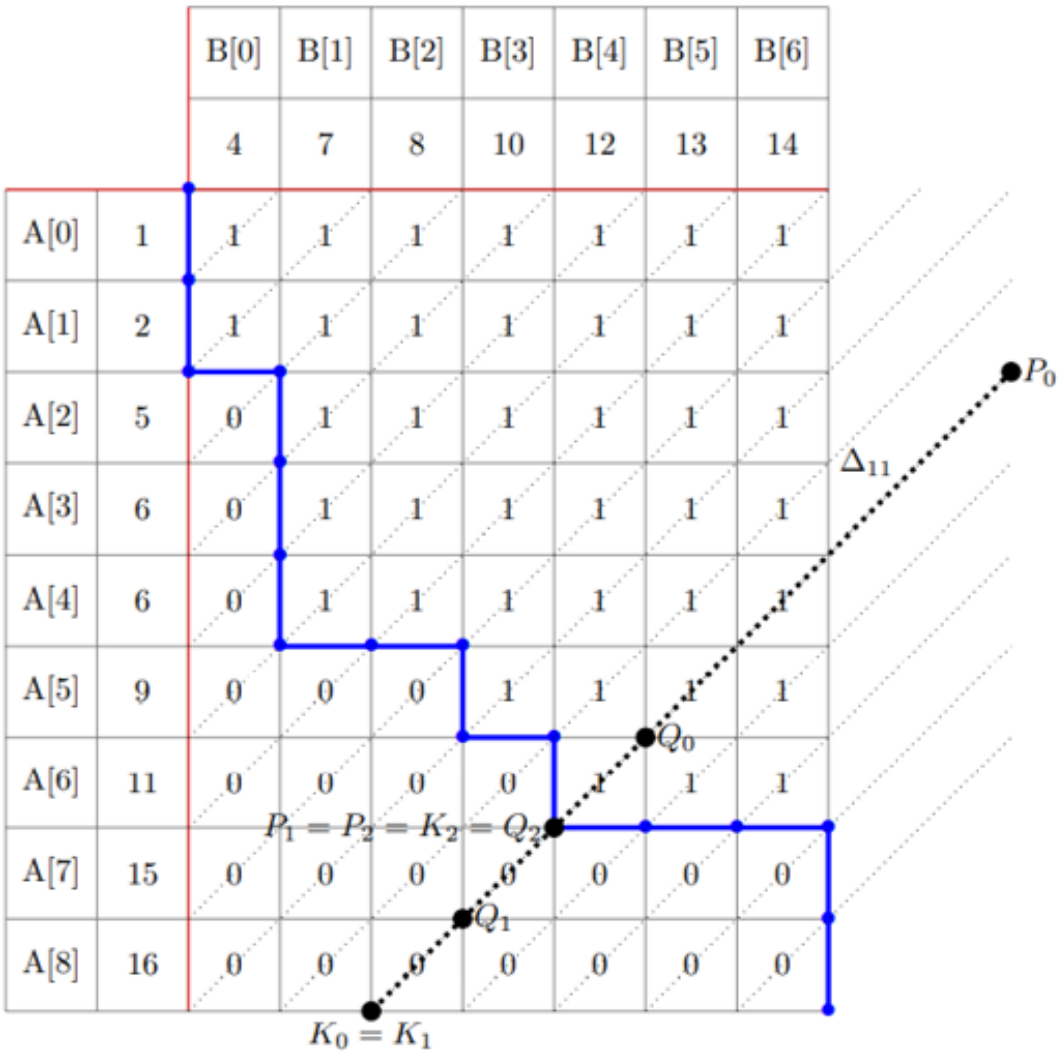

Figure – Exemple d'une procédure de Merge Path

Algorithme séquentiel

Algorithm 1 Sequential Merge Path

Require: A and B are two sorted arrays
Ensure: M is the merged array of A and B with $|M| = |A| + |B|$
  procedure MERGEPATH (A, B, M)
    $j = 0$ and $i = 0$
    while $i + j < |M|$ **do**
      if $i \geq |A|$ **then**
        M[i+j]=B[j]
        $j = j + 1$ ▷ The path goes right
      else if $j \geq |B|$ or $A[i] < B[j]$ **then**
        M[i+j]=A[i] ▷ The path goes down
        $i = i + 1$
      else
        M[i+j]=B[j]
        $j = j + 1$ ▷ The path goes right
      end if
    end while
  end procedure

Figure – Merge Path séquentiel

Algorithme GPU

Algorithm 2 Merge path (Indexes of n threads are 0 to $n-1$)

Require: A and B are two sorted arrays
Ensure: M is the merged array of A and B with $|M| = |A| + |B|$

```
for each thread in parallel do
    i=index of the thread
    if i > |A| then
        K = (i − |A|, |A|)                    ▷ Low point of diagonal
        P = (|A|, i − |A|)                    ▷ High point of diagonal
    else
        K = (0, i)
        P = (i, 0)
    end if
    while True do
        offset = abs(K_y − P_y)/2
        Q = (K_x + offset, K_y − offset)
        if Q_y ≥ 0 and Q_x ≤ B and
          (Q_y = |A| or Q_x = 0 or A[Q_y] > B[Q_x − 1]) then
          if Q_x = |B| or Q_y = 0 or A[Q_y − 1] ≤ B[Q_x] then
              if Q_y < |A| and (Q_x = |B| or A[Q_y] ≤ B[Q_x]) then
                  M[i] = A[Q_y]               ▷ Merge in M
              else
                  M[i] = B[Q_x]
              end if
              Break
          else
              K = (Q_x + 1, Q_y − 1)
          end if
        else
            P = (Q_x − 1, Q_y + 1)
        end if
    end while
end for
```

Figure – Merge Path parallélisé

Fusion ordonnée de deux listes déjà triées

Entrées :

- Vecteur A trié par ordre croissant et de taille d_A

$$ex : \quad [1, 2, 5, 6, 6, 9, 11, 15, 16]$$

- Vecteur B trié par ordre croissant et de taille d_B

$$ex : \quad [4, 7, 10, 12, 13, 14]$$

Sortie :

- Vecteur M trié par ordre croissant et de taille $d := d_A + d_B$

$$ex : \quad [1, 2, 4, 5, 6, 6, 7, 9, 10, 11, 12, 13, 14, 15, 16]$$

Fusion ordonnée de deux listes déjà triées

Détails sur l'implémentation de l'algorithme parallélisé :
La fusion est réalisée sur un seul bloc (voir consignes). Elle nécessite théoriquement d *threads*.

- Si $d <=$ NumThreadByBlock : Pas de problème.
- Si $d >$ NumThreadByBlock : Parcourir plusieurs fois les *threads*.

Fusion ordonnée de deux listes déjà triées

	Temps d'exécution (ms)
Sequential Merge Path	≈ 3.8
GPU Merge Path	≈ 6.4

Table – Temps d'exécution des algorithmes avec $d_A = 250000$ et $d_B = 500000$

Remarque :

- Dans ces conditions, l'algorithme séquentiel est plus efficace.
- On peut réduire le temps d'exécution de l'algorithme GPU Merge Path en augmentant le nombre de bloc de sorte que chaque *thread* ne soit utilisé qu'une seule fois. Alors, l'algorithme GPU Merge Path parvient à faire légérement mieux que l'algorithme séquentiel pour ces valeurs d_A et d_B.

Fusion ordonnée d'un batch de listes triées (deux par deux)

- **Entrées :** Vecteurs $(A)_{i=1}^{N}$ triés par ordre croissant et de taille d_A

$$ex : \quad [1, 2, 5, 6, 6, 9, 11, 15, 16]$$
$$[11, 12, 15, 16, 16, 19, 21, 25, 26]$$
$$[21, 22, 25, 26, 26, 29, 31, 35, 36]$$

- **Entrées :** Vecteurs $(B)_{i=1}^{N}$ triés par ordre croissant et de taille d_B

$$ex : \quad [4, 7, 10, 12, 13, 14]$$
$$[14, 17, 20, 22, 23, 24]$$
$$[24, 27, 20, 32, 33, 34]$$

- **Sortie :** Vecteurs $(M)_{i=1}^{N}$ triés par ordre croissant et de taille $d := d_A + d_B$

$$ex : \quad [1, 2, 4, 5, 6, 6, 7, 9, 10, 11, 12, 13, 14, 15, 16]$$
$$[11, 12, 14, 15, 16, 16, 17, 19, 20, 21, 22, 23, 24, 25, 26]$$
$$[21, 22, 24, 25, 26, 26, 27, 29, 30, 31, 32, 33, 34, 35, 36]$$

Fusion ordonnée d'un batch de listes triées (deux par deux)

Détails sur l'implémentation de l'algorithme parallélisé :
Chaque fusion est réalisée sur un seul bloc (voir consignes). Il faut donc définir N blocs.

- Si $d <=$ NumThreadByBlock : Pas de problème.
- Si $d >$ NumThreadByBlock : Parcourir plusieurs fois les *threads* de chaque bloc.

Fusion ordonnée d'un batch de listes triées (deux par deux)

	Temps d'exécution (ms)
Sequential Batch Merge Path	≈ 4.4
GPU Batch Merge Path	≈ 2.0

Table – Temps d'exécution des algorithmes avec $N = 100$, $d_A = 2500$ et $d_B = 5000$

Remarque :

- Dans ces conditions, l'algorithme GPU Batch Merge Path est plus efficace.

Conclusion

- Lorsque le GPU est utilisé de manière optimale, l'algorithme de l'article est bien plus rapide que l'équivalent séquentiel (d'autant plus lorsque N est grand).
- Pour aller plus loin, nous pourrions répéter plusieurs fois les expériences précédentes afin de réaliser des statistiques sur les temps d'éxécution. Nous pourrions également optimiser encore d'avantage l'utilisation du GPU pour une mise en production à grande échelle.

Bibliographie


Oded Green, Robert McColl, and David A. Bader.
Gpu merge path : A gpu merging algorithm.
In *Proceedings of the 26th ACM International Conference on Supercomputing*, ICS '12, page 331–340, New York, NY, USA, 2012. Association for Computing Machinery.