

## Projet : Batch merge and merge path sort

Code source et présentation du travail le 10/12/2020

Le code source doit tre lisible et suffisamment comment

La dure des présentations est de  $\sim 15$  minutes par binme

Based on the merging sort presented in [1], this project has three parts. The first part deals with sorting an array on GPU. The second part deals with batch merging various small arrays at the same time. The last part is left to students to propose applications involving merging arrays. In the following, given a set  $A$ , the cardinal  $|A|$  denotes the number of its elements.

### 1 Merge path and sort

We start with the merge path algorithm. Let  $A$  and  $B$  be two ordered arrays (increasing order), we want to merge them in an  $M$  sorted array. The merge of  $A$  and  $B$  is based on a path that starts at the top-left corner of the  $|A| \times |B|$  grid and arrives at the down-right corner. The Sequential Merge Path is given by Algorithm 1 and an example is provided in Figure 1.

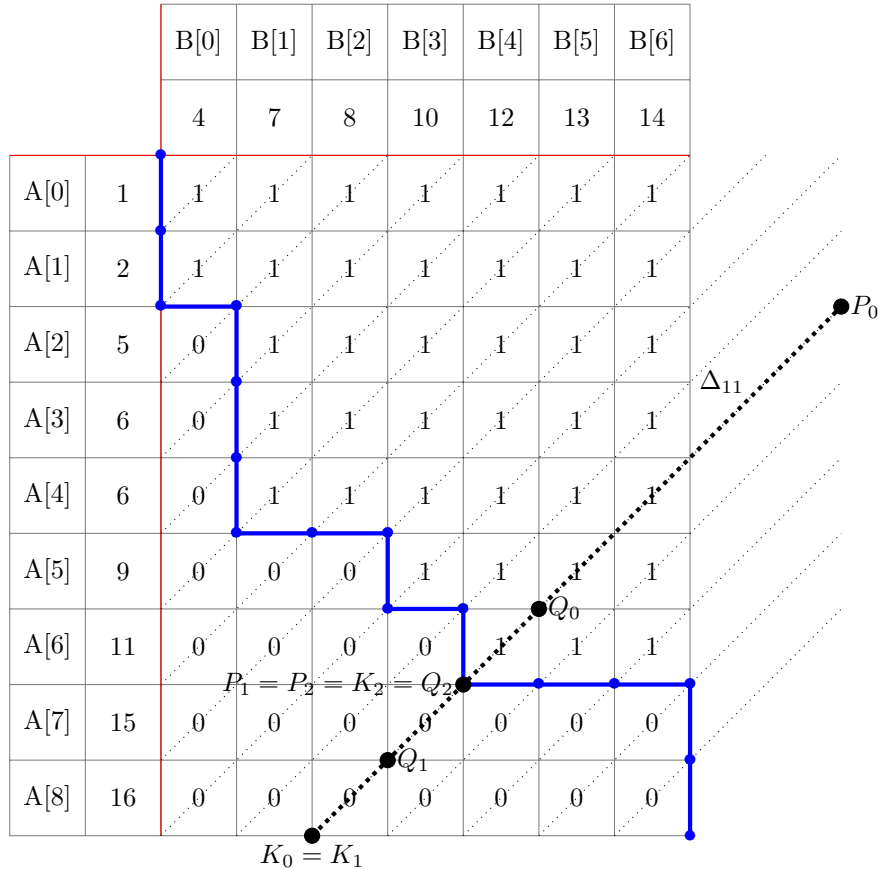


Figure 1: An example of Merge Path procedure

---

**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**

$M[i+j] = A[i]$

$i = i + 1$

▷ The path goes down

**else**

$M[i+j] = B[j]$

$j = j + 1$

▷ The path goes right

**end if**

**end while**

**end procedure**

---

---

**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**

$i = \text{index of the thread}$

**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**

$offset = \text{abs}(K_y - P_y)/2$

$Q = (K_x + offset, K_y - offset)$

**if**  $Q_y \geq 0$  and  $Q_x \leq B$  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  $A[Q_y - 1] \leq B[Q_x]$  **then**

**if**  $Q_y < |A|$  and  $(Q_x = |B|$  or  $A[Q_y] \leq B[Q_x])$  **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**

---

Each point of the grid has a coordinate  $(i, j) \in \llbracket 0, |A| \rrbracket \times \llbracket 0, |B| \rrbracket$ . The merge path starts from the

point  $(i, j) = (0, 0)$  on the left top corner of the grid. If  $A[i] < B[j]$  the path goes down else it goes right. The array  $\llbracket 0, |A| - 1 \rrbracket \times \llbracket 0, |B| - 1 \rrbracket$  of boolean values  $A[i] < B[j]$  is not important in the algorithm. However, it shows clearly that the merge path is a frontier between ones and zeros.

To parallelize the algorithm, the grid has to be extended to the maximum size equal to  $\max(|A|, |B|) \times \max(|A|, |B|)$ . We denote  $K_0$  and  $P_0$  respectively the low point and the high point of the ascending diagonals  $\Delta_k$ . On GPU, each thread  $k \in \llbracket 0, |A| + |B| - 1 \rrbracket$  is responsible of one diagonal. It finds the intersection of the merge path and the diagonal  $\Delta_k$  with a binary search described in Algorithm 2.

1. For  $|A| + |B| \leq 1024$ , write a kernel `mergeSmall.k` that merges  $A$  and  $B$  using only one block of threads.
2. For any size  $|A| + |B| = d$  sufficiently smaller than the global memory, write two kernels that merge  $A$  and  $B$  using various blocks: The first kernel `pathBig.k` finds the merge path and the second one `mergeBig.k` merges  $A$  and  $B$ .
3. Looping on appropriate calls of `pathBig.k` and of `mergeBig.k`, write a function that sorts any array  $M$  of size  $d$  sufficiently smaller than the global memory. Give the execution time with respect to  $d$ .

## 2 Batch merge

In this part, we assume that we have a large number  $N (\geq 1e3)$  of arrays  $\{A_i\}_{1 \leq i \leq N}$  and  $\{B_i\}_{1 \leq i \leq N}$  with  $|A_i| + |B_i| = d \leq 1024$  for each  $i$ . Using some changes on `mergeSmall.k`, we would like to write `mergeSmallBatch.k` that merges two by two, for each  $i$ ,  $A_i$  and  $B_i$ .

Given a fixed common size  $d \leq 1024$ , `mergeSmallBatch.k` is launched using the syntax

```
mergeSmallBatch.k<<<numBlocks, threadsPerBlock>>>(...);
```

with `threadsPerBlock` is multiple of  $d$  but smaller than 1024 and `numBlocks` is an arbitrary sufficiently big number.

4. Explain why the indices  

```
int tid = threadIdx.x % d;
```

```
int Qt = (threadIdx.x - tid) / d;
```

```
int gbx = Qt + blockIdx.x * (blockDim.x / d);
```

are important in the definition of `mergeSmallBatch.k`.
5. Write the kernel `mergeSmallBatch.k` that batch merges two by two  $\{A_i\}_{1 \leq i \leq N}$  and  $\{B_i\}_{1 \leq i \leq N}$ . Give the execution time with respect to  $d = 4, 8, \dots, 1024$ .

## 3 Merge sort applications

You are free to give a specific application of either question 3. or question 5.

## References

- [1] O. Green, R. McColl and D. A. Bader GPU Merge Path - A GPU Merging Algorithm. *26th ACM International Conference on Supercomputing (ICS)*, San Servolo Island, Venice, Italy, June 25-29, 2012.