Sorbonne Université

Calcul haute performance : programmation et algorithmique avancées

Projet : Merge Sort
Code source et présentation du travail le 01/12/2022

Le code source doit être lisible et suffisamment commenté
La durée des présentations est de $\sim$ 18 minutes par binôme + 5 minutes de questions

Based on the merging sort presented in [1], this project has three parts. The first part is about merging two sorted arrays using one block of threads. The second part deals with batch merging and thus sorting small arrays. In the last part, students will be able to merge sort any array using multiple blocks. In the following, given a set A, the cardinal $|A|$ denotes the number of its elements. **Any recursive call should be avoided.**

1 Merge path (/10 for MAIN5+SAR and /8 for SFPN)

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.
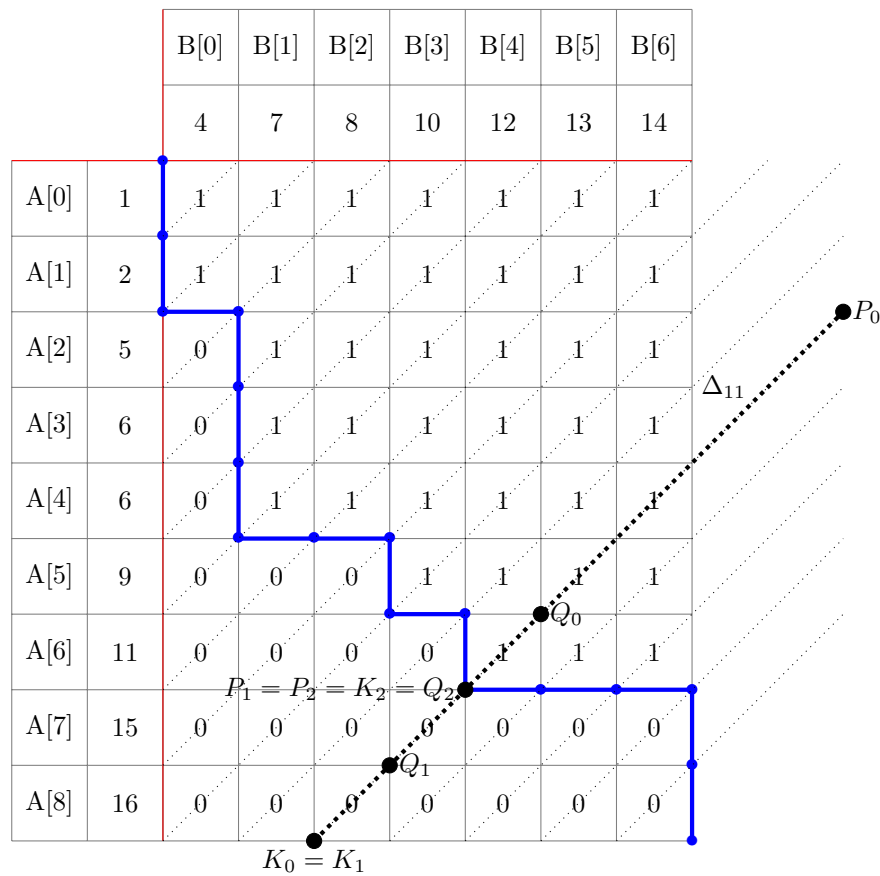

Figure 1: An example of Merge Path procedure

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]   ▷ 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

Algorithm 2 Merge Path (Indices 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 $Q_y \geq 0$ 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 $(i, j) \in [\![0, |A|]\!] \times [\![0, |B|]\!]$. 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 $[\![0, |A| - 1]\!] \times [\![0, |B| - 1]\!]$ 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 Δ_k. On GPU, each thread $k \in [\![0, |A| + |B| - 1]\!]$ is responsible of one diagonal. It finds the intersection of the merge path and the diagonal Δ_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. (/6 or /5)

2. Use this same kernel to merge various arrays of As and Bs independently on multiple blocks and study the execution time with respect the size of $|A| + |B|$. (/4 or /3)

2 Batch merge and sort (/6 for MAIN5+SAR and /5 for SFPN)

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.

3. Explain why the indices
   ```
   int tidx = threadIdx.x%d;
   int Qt = (threadIdx.x-tidx)/d;
   int gbx = Qt + blockIdx.x*(blockDim.x/d);
   ```
   are important in the definition of `mergeSmallBatch_k`.
   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 $d = 4, 8, ..., 1024$. (/4)

We assume now that we have a large number $N(\geq 1e3)$ of arrays $\{M_i\}_{1 \leq i \leq N}$ with $|M_i| = d \leq 1024$ and we would like to write `SortSmall_k` that sorts each M_i.

4. Define the kernel `SortSmall_k` and test it for multiple values of d. (/2 or /1)

3 Merge sort (/4 for MAIN5+SAR and /3 for SFPN)

As mentioned in [1], merge path algorithm is divided into 2 stages: partitioning stage and merging stage. The partitioning stage is important to propose an algorithm that involves various blocks.

5. For any size $|A| + |B| = d$ sufficiently smaller than the global memory, write a solution that merges A and B using various blocks. (/2 or /1)

6. Write the final solution that sorts any array M of size d sufficiently smaller than the global memory. (/1)

7. Optimize with respect to the number of threads that should be used and study the execution time with respect to d. (/1)

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.