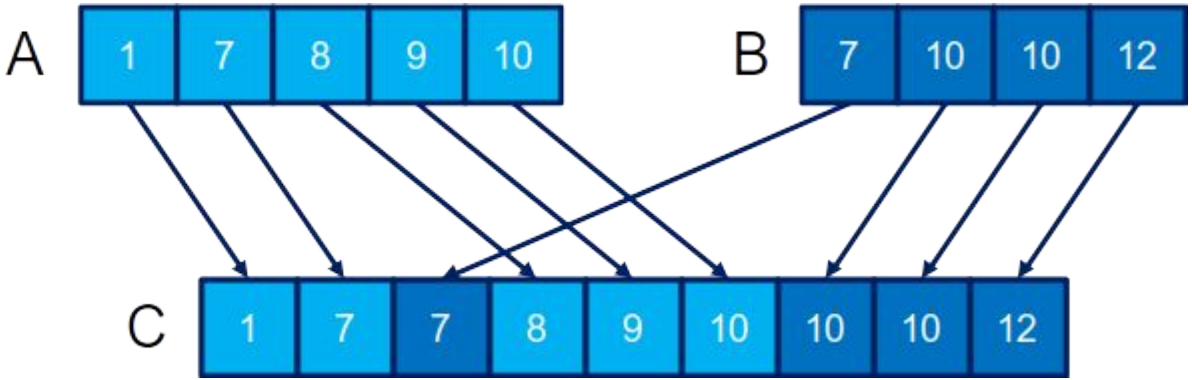
Merge

Dynamic input data identification

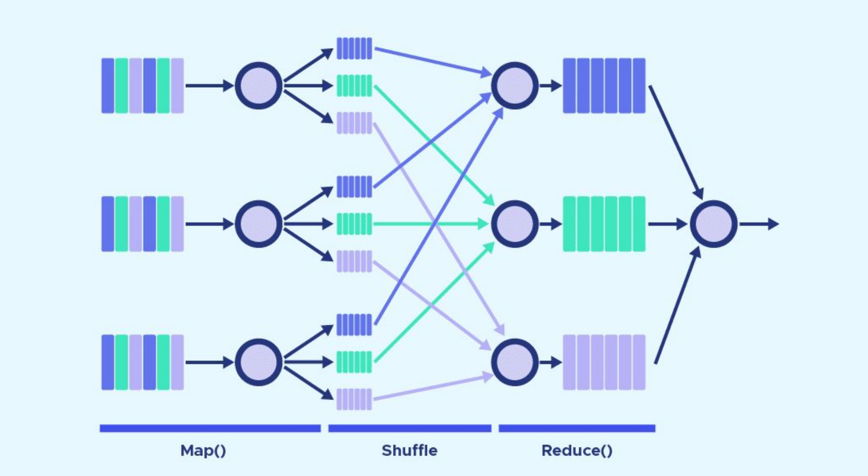
Merge Operation

- It takes two sorted lists and generates a combined sorted list
  - An ordered merge function takes two sorted lists, A and B, and merges them into a single sorted list, C
  - There is an order relation (e.g., less than or equal to) in the sorted lists and the merged list
- Focus on Stable Sort
  - Whenever the numerical values are equal between an element of A and an element of B, the element of A should appear first in the output list C
  - This requirement ensures the stability of the ordered merge operation.



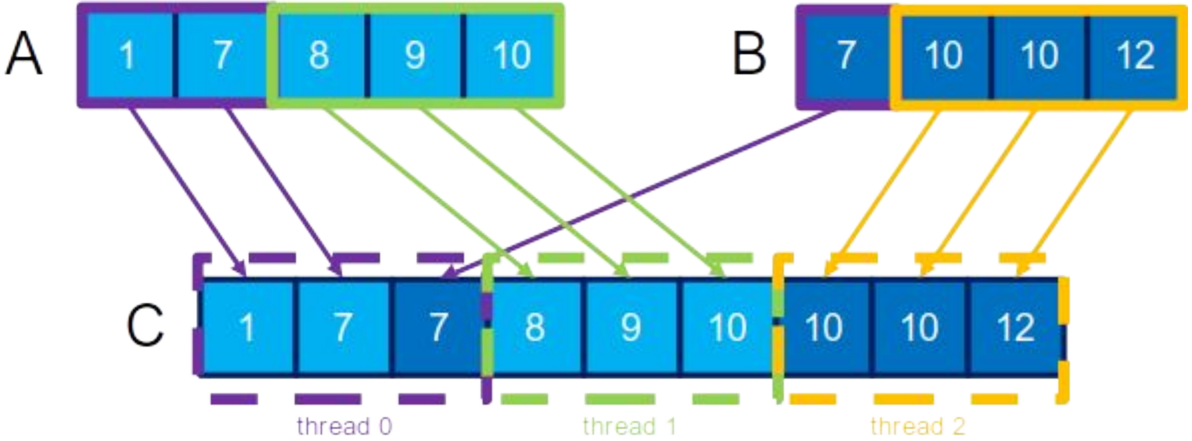
Why does it matter?

- It is a building block of modern mapreduce frameworks
  - For example, Hadoop distributes the computation over a large number of nodes
  - The reduce process assembles the result of these compute nodes into the final result
  - Results may be sorted according to an ordering relation.



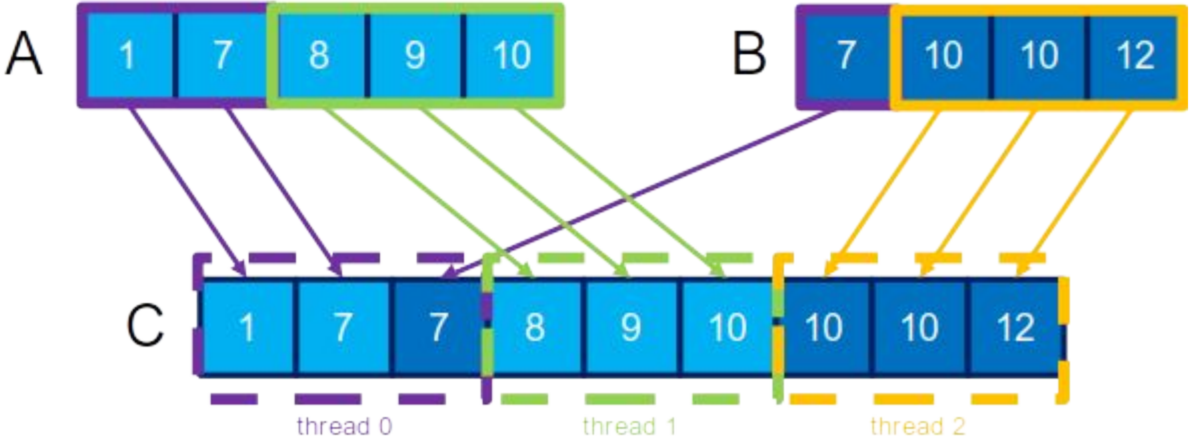
Naive Implementation - 1

- Sequential approach has $O(m+n)$ complexity
  - Where m and n are A and B dimensions
- We can gather the merge computation among threads
  - We partition the output list equally among threads
  - Each thread collects input elements for its section of the output
  - The range of input elements to be used by each thread is a function of the input elements



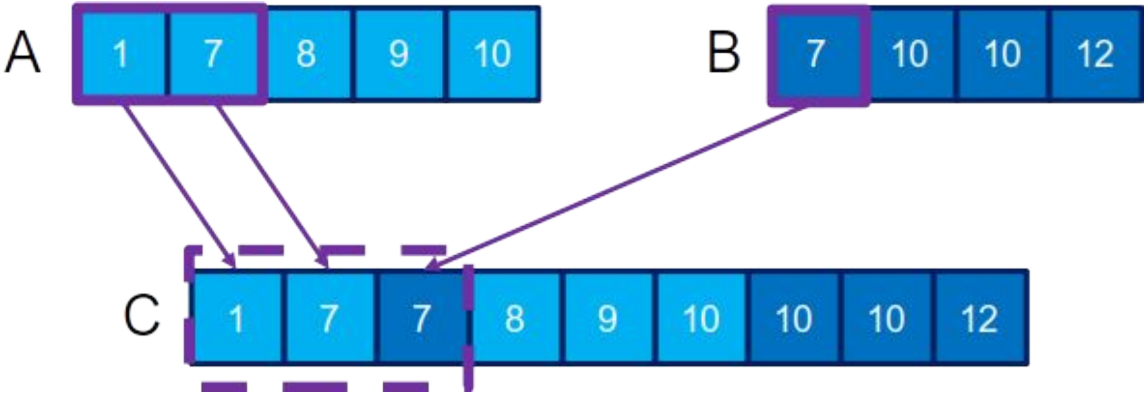
Naive Implementation - 2

- We can gather the merge computation among threads
  - Each output section receives its elements from a continuous section of A and B
- All threads identify the starting and ending locations of the continuous sections of the inputs (A and B) that they will use
  - All threads perform merge for their sections in parallel
  - Each thread executes a sequential merge for its own section

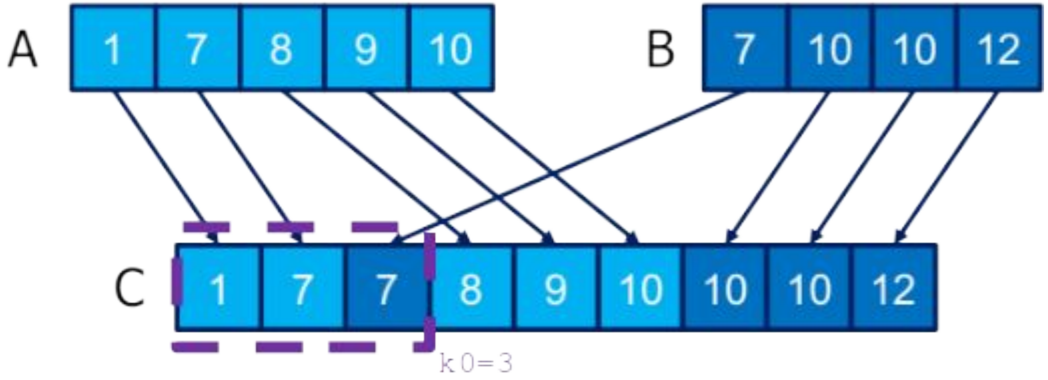


Co-Rank Calculation

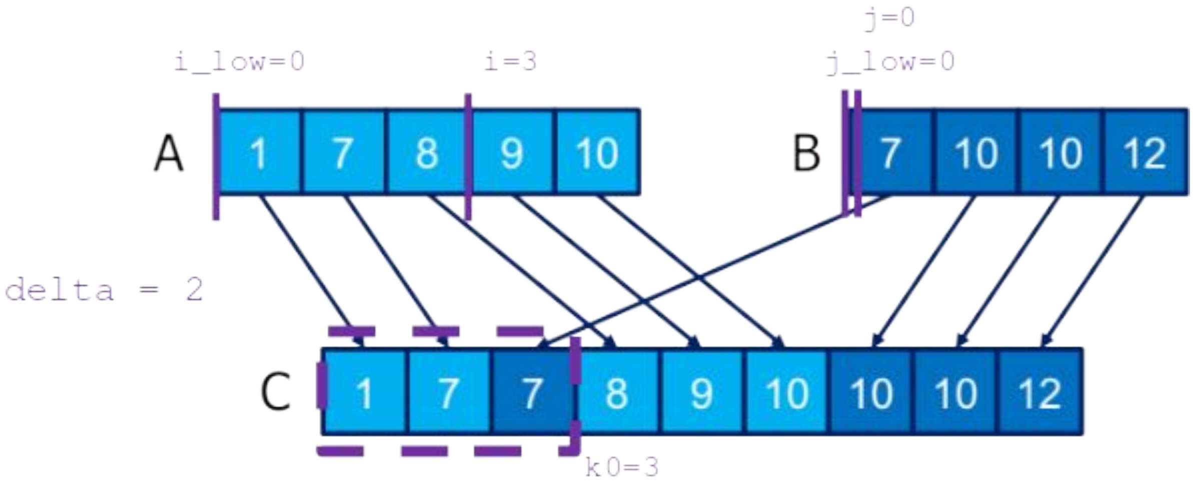
- For an element $C[k]$, k is referred to as its rank, and i and j are referred to as its <u>co-ranks</u>
  - For any k such that $0 \leq k < m+n$, we can find i and j such that $k=i+j$, $0 \leq i < m$ and $0 \leq j < n$
    - $A[i-1] \leq B[j]$
    - $B[j-1] < A[i]$
- Finding co-rank values for different threads is not balanced
- Use binary search for i and j values
  - Complexity of $O(\log N)$



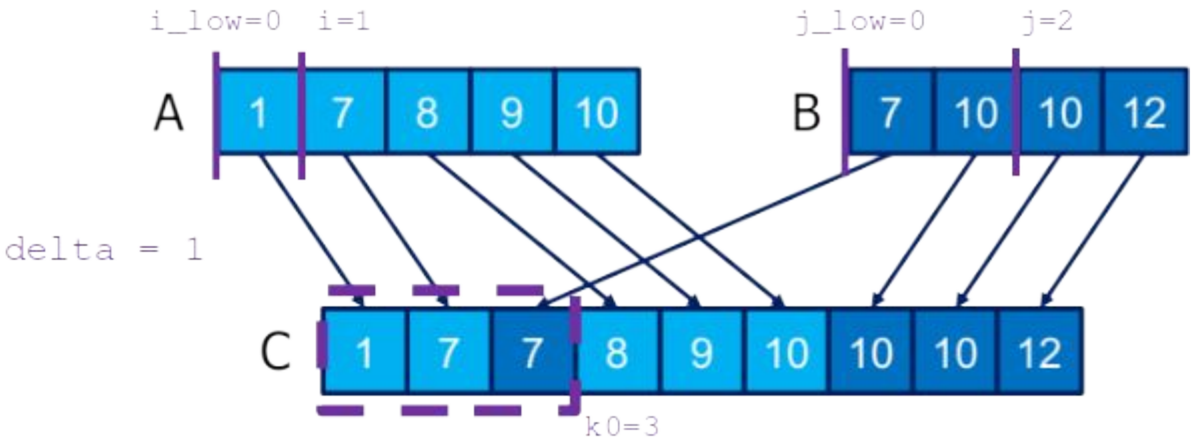
Co-Rank Example



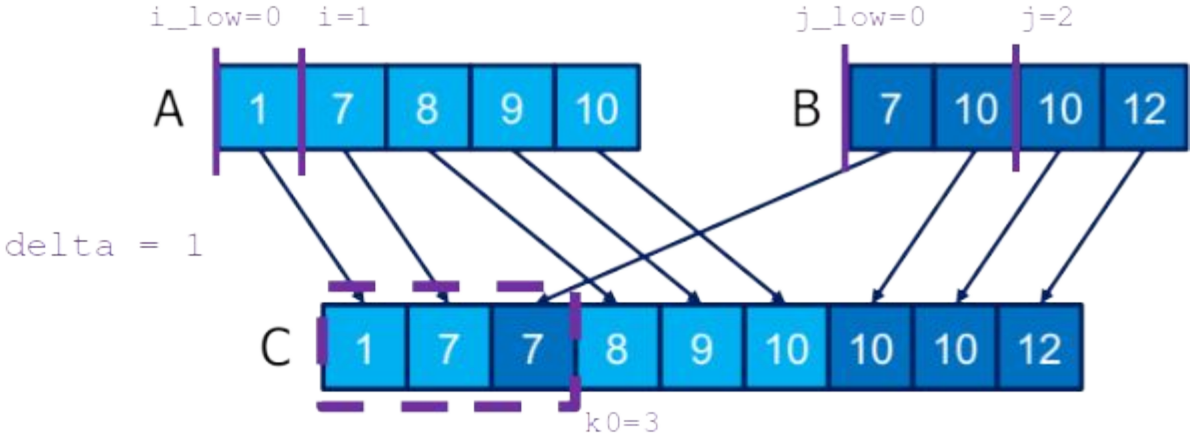
Co-Rank Example



Co-Rank Example

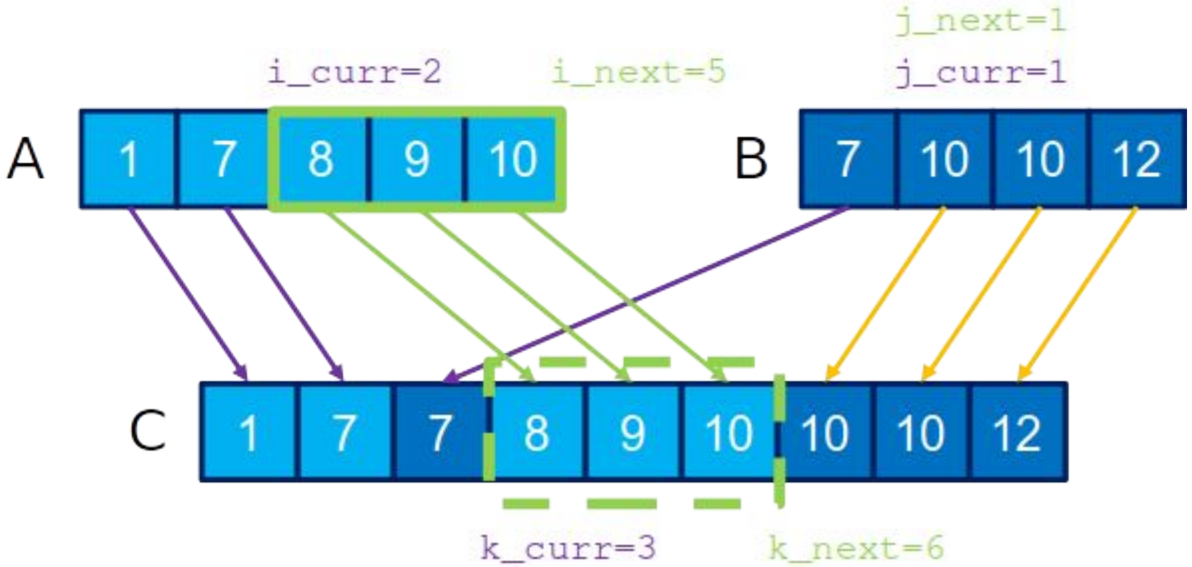


Co-Rank Example

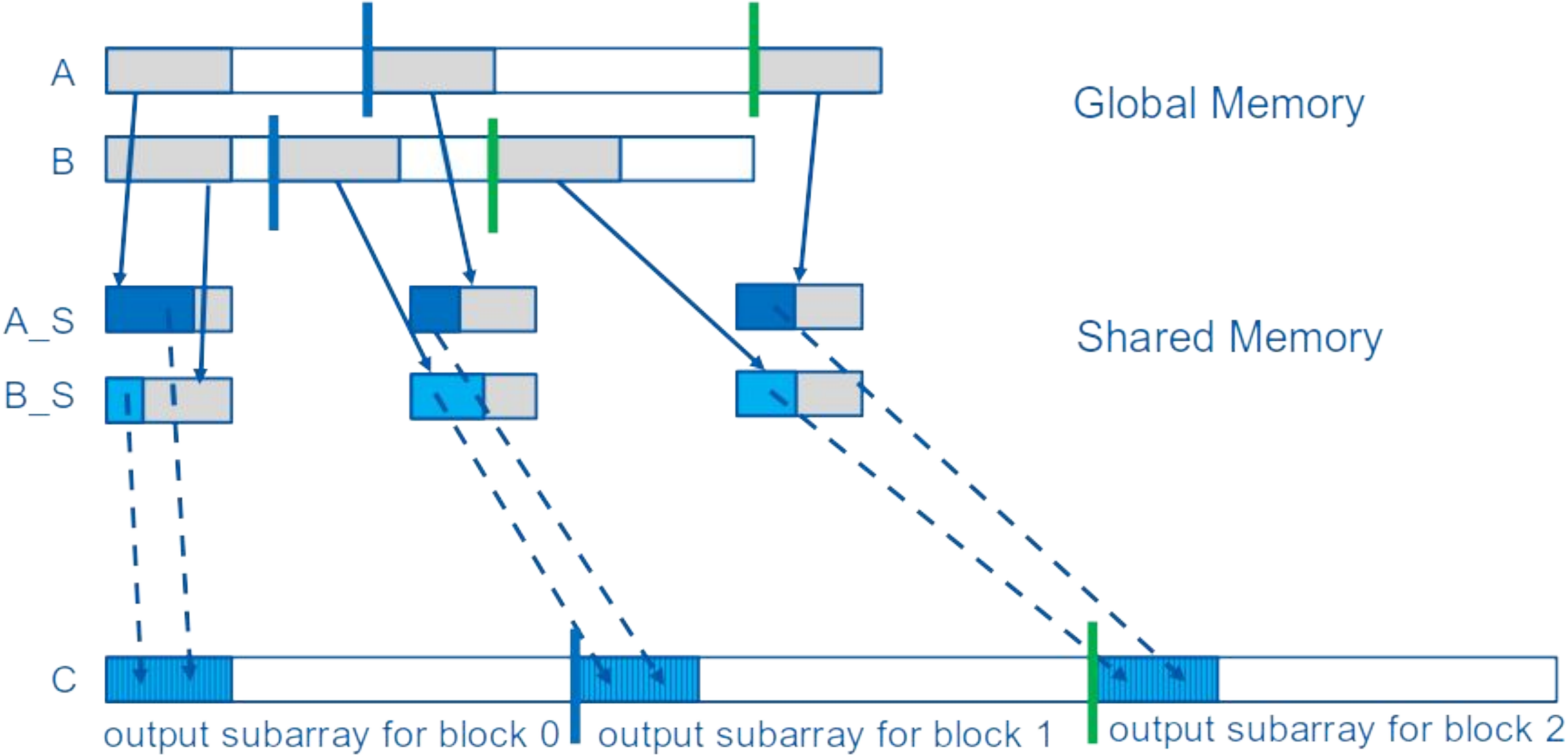


Parallel Merge

- Each thread is in charge of a continuous section of the output
- Problem
  - Uncoalesced memory accesses
  - We can improve the solution by collaborative loading sections of A and B into shared memory



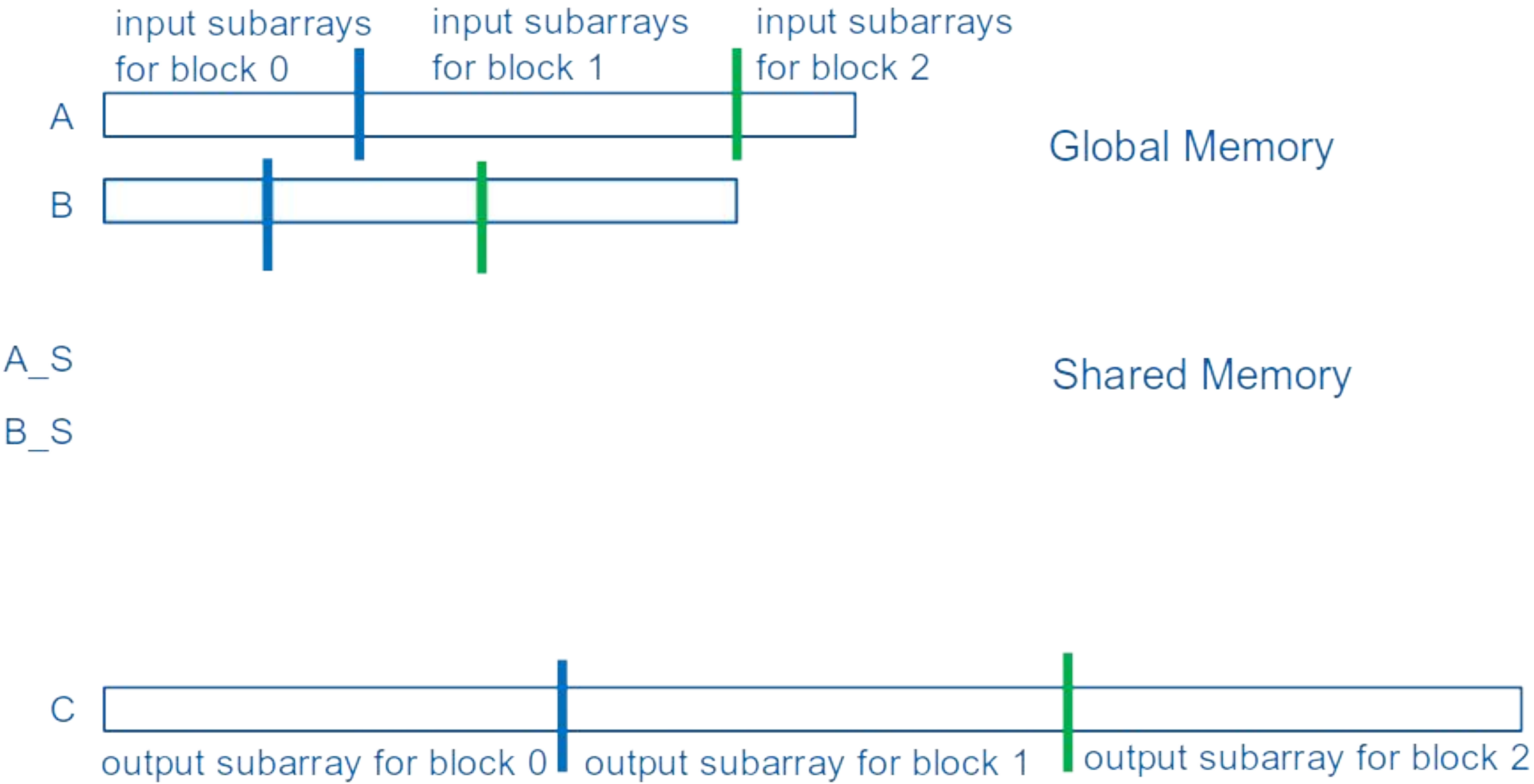
Tiled Merge


A
B
Global Memory
A_S
B_S
Shared Memory
C
output subarray for block 0
output subarray for block 1
output subarray for block 2

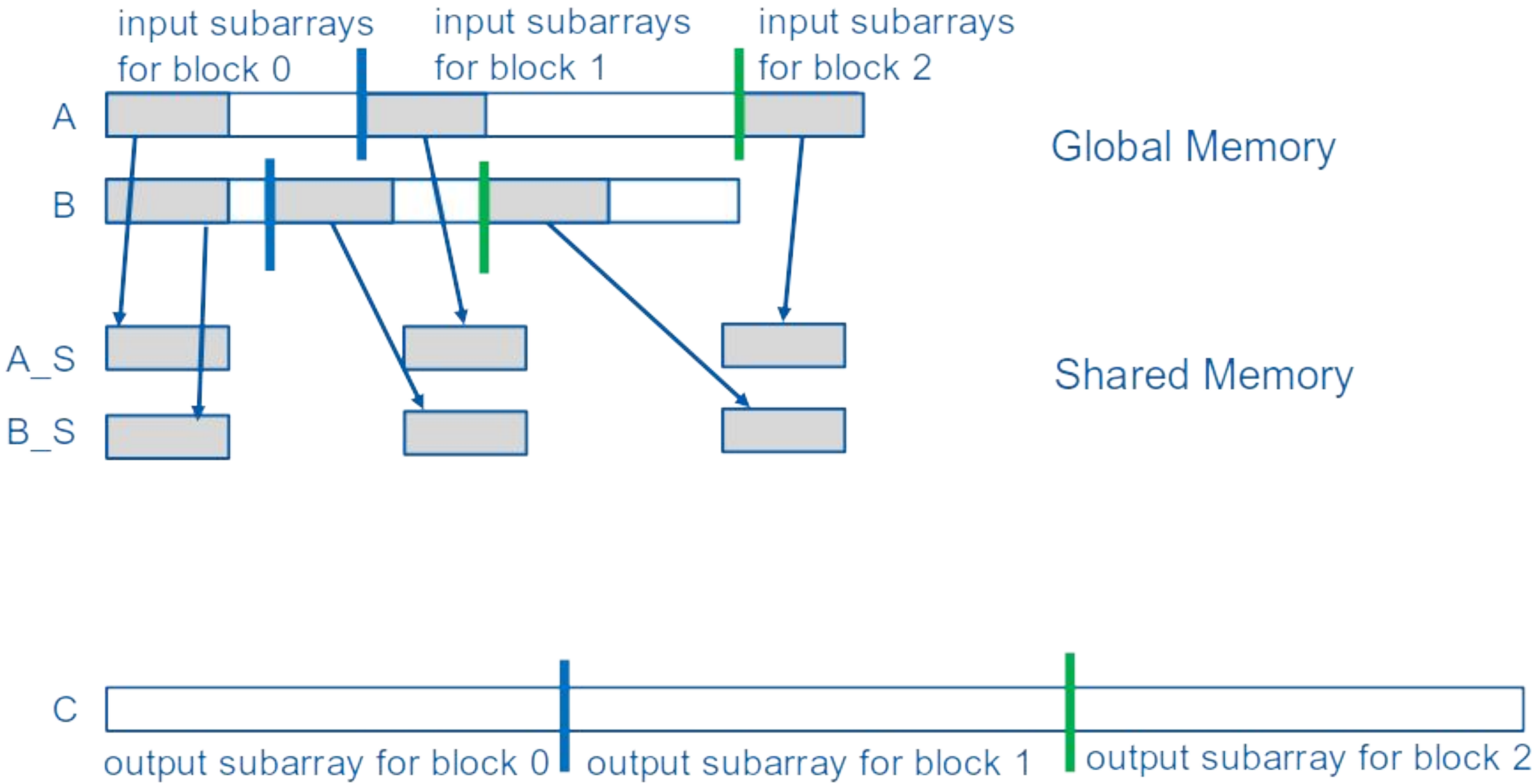
Tiled Merge

- Each thread block computes an output sections
  - It should be a multiple of the $TILE\_SIZE$ in the exercise
- The leader thread performs the binary search to identify the input sections
- Each block iteratively generates its output section; at each iteration
  - Threads of a block collaboratively load a tile of A and a tile of B into shared memory
  - Divide the output tile into subsections and assign them to threads
  - All threads perform parallel merge on input and output tiles

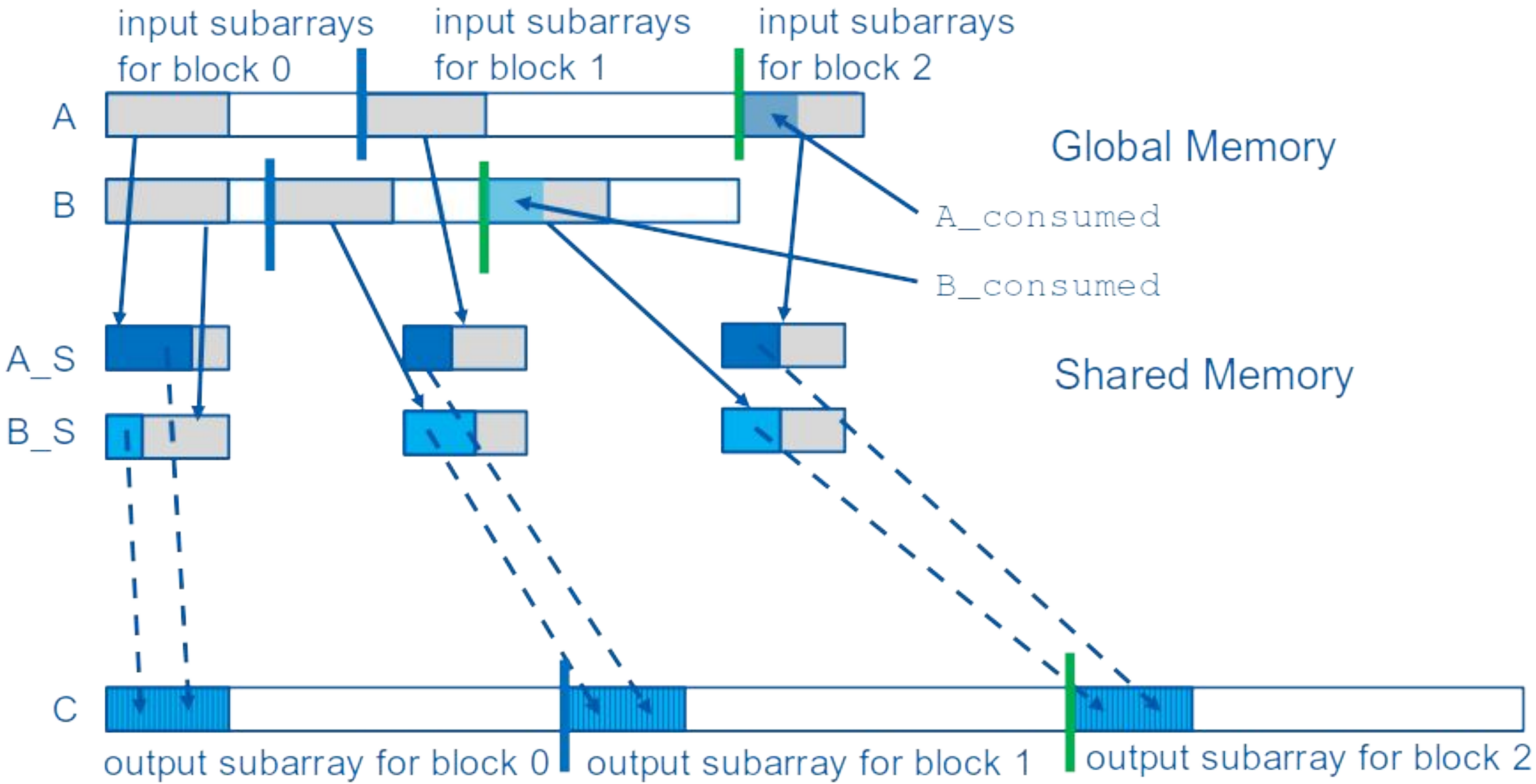
Tiled Merge Example (Iteration 0)



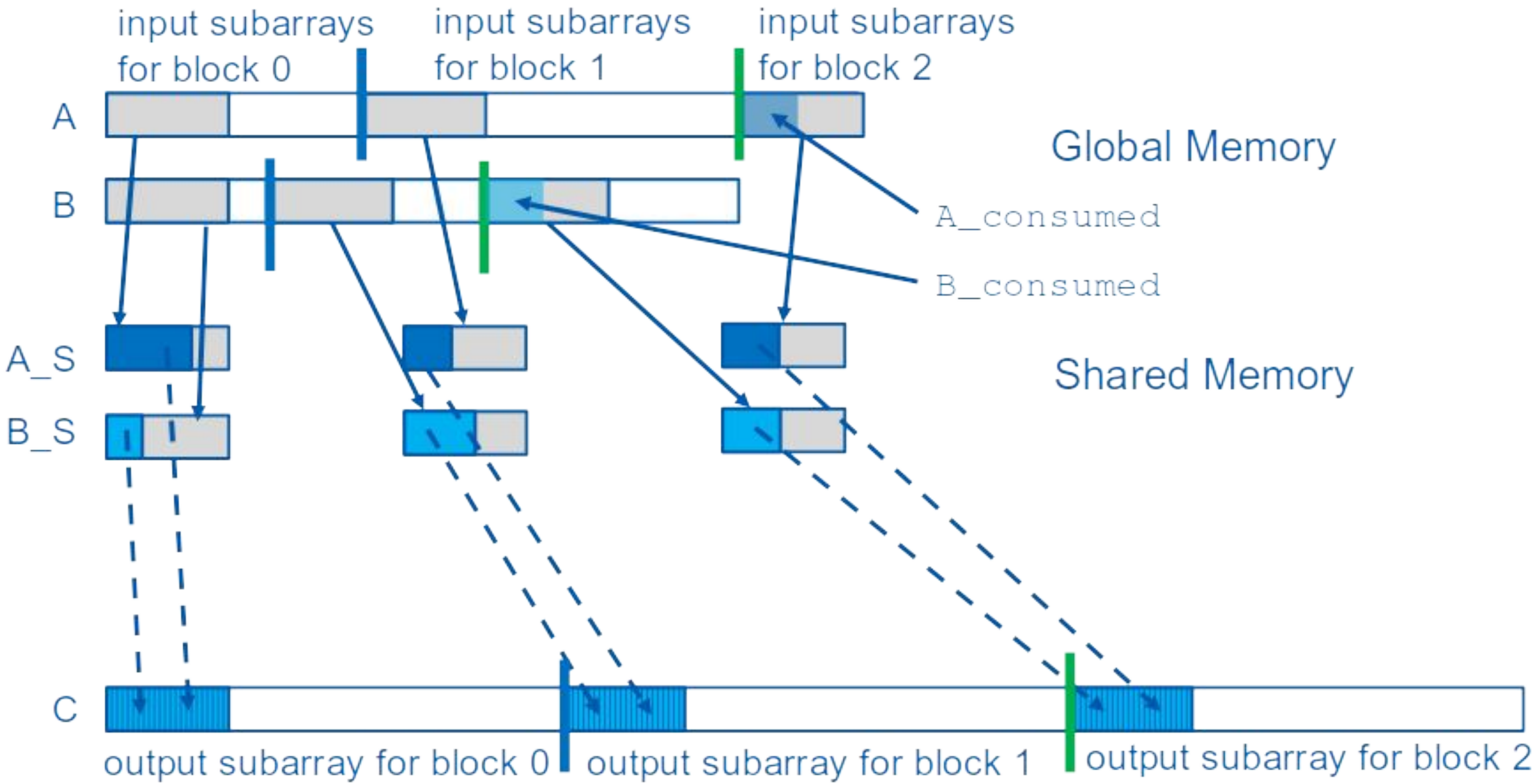
Tiled Merge Example (Iteration 0)



Tiled Merge Example (Iteration 0)



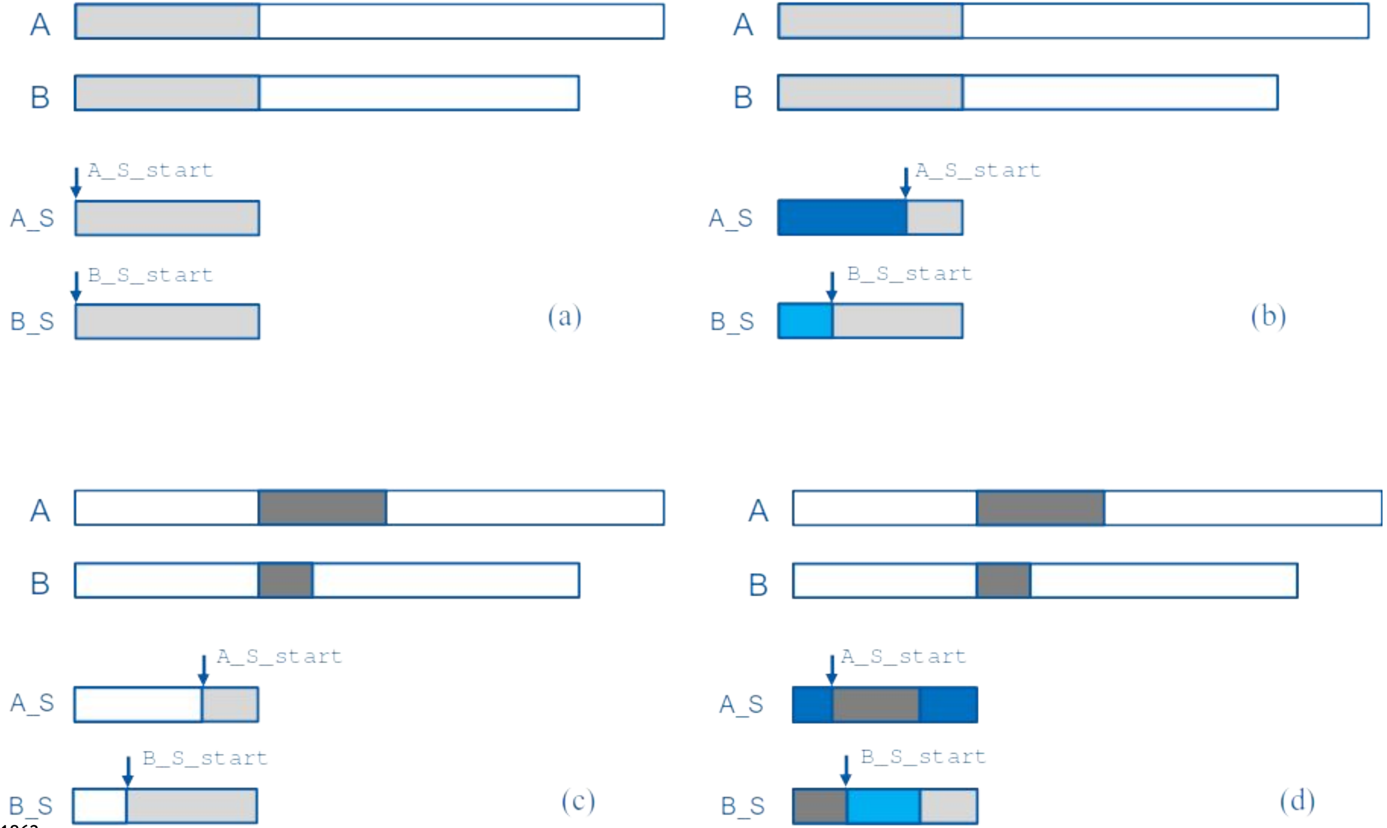
Tiled Merge Example (Iteration 1)



Tile Merge

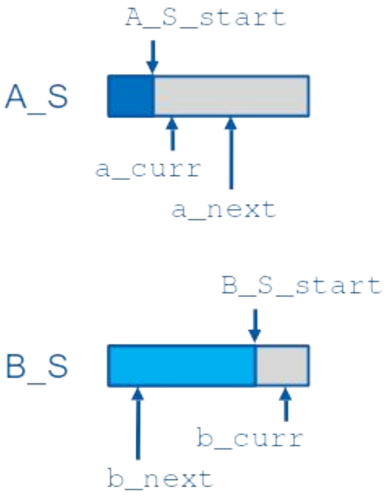
- Coalesced Access
- Reduced global memory access on co-rank functions
- Improvement
  - Coalesced stores if results are written to shared memory
  - Call the co_rank twice during initialization to get this, and the next output section dimensions
    - Thus, being more accurate in loading data
  - Only half of the input elements loaded into shared memory are used (in the worst case)

Circular Buffering

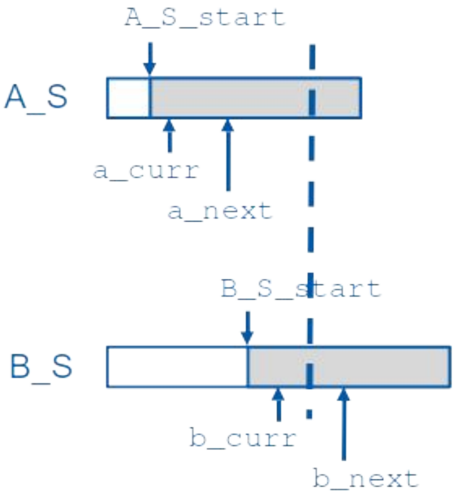


Circular Buffering: Co-Rank Values

- Handling the co-rank computation is now more complex



(a) Reality



(b) Simplified

Credits

- PUMPS+AI 2022 Lecture 2: Input Regularization Techniques for Gather Parallelization, from *Wen-mei Hwu, Mateo Valero, Toni Pena, Juan Gomez-Luna, Marc Jorda, Leonidas Kosmidis, Bernat Font*

Thank you for your attention!

Gianmarco Accordi
gianmarco.accordi@polimi.it