1. Let $X = k_1, k_2, \ldots, k_n$. Assume without loss of generality that the keys are distinct. Note that the right neighbor of any input key $k_i$ is nothing but the minimum among all the input keys that are greater than $k_i$. Key $k_i$ is assigned a group $G_i$ of $n$ processors, $1 \leq i \leq n$. The processors associated with $k_i$ use an array $A_i[1 : n]$. This array is initialized with all $\infty$’s. Processor $j$ of group $G_i$ writes $k_j$ in $A_i[j]$ if $k_j > k_i$. After this write step that takes one parallel step, processors in $G_i$ find the minimum of $A_i[1], A_i[2], \ldots, A_i[n]$ in $O(1)$ time. This minimum is the right neighbor of $k_i$.

2. We will show that we can stably sort $n$ integers in the range $[1, \sqrt{n}]$ in $O(\sqrt{n})$ time using $\sqrt{n}$ CREW PRAM processors. Using the idea of radix sorting it will follow that we can sort $n$ integers in the range $[1, n^c]$ (for any constant $c$) in $O(\sqrt{n})$ time using $\sqrt{n}$ processors.

Let $X = k_1, k_2, \ldots, k_n$ be the input sequence. Assign $\sqrt{n}$ keys per processor. In particular, the first processor gets the keys $k_1, k_2, \ldots, k_{\sqrt{n}}$; the second processor gets the keys $k_{\sqrt{n}+1}, k_{\sqrt{n}+2}, \ldots, k_{2\sqrt{n}}$; and so on.

(a) Each processor sorts its keys using bucket sorting. This takes $O(\sqrt{n})$ time. Let $N_{i,j}$ be the number of keys of value $j$ that processor $i$ has, for $1 \leq i, j \leq \sqrt{n}$.

(b) All the $\sqrt{n}$ processors perform a prefix sums computation on $N_{1,1}, N_{2,1}, \ldots, N_{\sqrt{n},1}$, $N_{1,2}, N_{2,2}, \ldots, N_{\sqrt{n},2}$, $\ldots$, $N_{1,\sqrt{n}}, N_{2,\sqrt{n}}, \ldots, N_{\sqrt{n},\sqrt{n}}$.

(c) Each processor now uses these prefix sums values to output its keys in the sorted order.

Since each of the above three steps takes $O(\sqrt{n})$ time, the run time of the algorithm is $O(\sqrt{n})$.

3. Assume that $A$ and $B$ are in common memory in successive cells. In particular, assume that $A$ is in $M[1 : n]$ and $B$ is in $M[n + 1 : m + n]$.

(a) Sort $B$, i.e., sort $M[n + 1 : n + m]$. This can be done in $O(\log m)$ time using $m$ arbitrary CRCW PRAM processors.

(b) Assign one processor per element of $A$. Processor $i$ performs a binary search in $B[n + 1 : n + m]$ to check if $M[i]$ is in $B$, for $1 \leq i \leq n$. This binary search takes $O(\log m)$ time.

(c) In this step, we’ll use an array $Q[1 : 2m]$. Each element of $A$ that is also in $B$ will be placed in a unique cell of $Q$. Each element of $A$ is assigned one processor. If an element of $A$ is in $A \cap B$, the corresponding processor will try to place the element in $Q$. If an element of $A$ is not in $A \cap B$, the corresponding processor goes to sleep. If a processor $\pi$ has an element that has to be placed in $Q$, $\pi$ proceeds in rounds. It takes as many rounds as needed to successfully place its key.

In a round, $\pi$ picks a random cell in $Q$: If this cell is occupied, it waits for the next round; If this cell is empty, it tries to write its key in the cell; Processor $\pi$ reads from this cell to check if its key is there; If so, the processor goes to sleep; If not, it moves to the next round.
Probability that \( \pi \) succeeds in any round is \( \geq 1/2 \). Thus the number of rounds needed to place \( \pi \)'s key successfully in \( Q \) is \( \tilde{O}(\log m) \), for any processor \( \pi \).

(d) Use a prefix computation to compress the array \( Q[1 : 2m] \) (and get rid of the empty cells). This can be done in \( O(\log m) \) time using \( \frac{2m}{\log m} \leq n \) processors.

The compressed array \( Q \) is \( A \cap B \).

We could do steps (c) and (d) in a different way as follows. We use an array \( Q[1 : m] \) initialized to all zeros. Each element of \( A \) is assigned a processor. Processor \( i \) goes to sleep if \( k_i \) is not in \( A \cap B, 1 \leq i \leq n \). Otherwise, processor \( i \) writes a 1 in \( Q[j] \) if \( M[i] = M[n + j] \). After this parallel write step, we assign one processor per element of \( B \). These processors empty the cells of \( B \) that are not in \( A \cap B \). A prefix sums computation is done on \( Q \) in \( O(\log m) \) time using \( \frac{m}{\log m} \) processors. These prefix sums are used to write the elements of \( A \cap B \) in successive cells in common memory.