4-WAY RADIX SORTING IMPLEMENTATION IN CUDA

M. VENKATAKRISHNA

KEY-DIFFERENCE 1: COMPACT ARRAYS

- The implementation of 4-Way radix sort is similar to 2-way radix sort. **There are only these key differences**:

```
//Takes arr[] as input and produces compact_0[] for bit-00, compact_1[] for bit-01, compact_2[] for bit-10 and compact_3[] for bit-11 , in
__global__ void compacter(unsigned int* arr, unsigned int* compact_0, unsigned int* compact_1,
                          unsigned int* compact_2, unsigned int* compact_3, int i){ ... }
```

❑ The compacter() kernel function is now **responsible for producing 4 compact[] arrays** :
compact_0[] , compact_1[], compact_2[] and compact_3[].

- **compacter_0[k] == 1** if arr[k] has **(bit-i , bit-i+1) as (“00”)**
- **compacter_1[k] == 1** if arr[k] has **(bit-i , bit-i+1) as (“01”)**
- **compacter_2[k] == 1** if arr[k] has **(bit-i , bit-i+1) as (“10”)**
- **compacter_3[k] == 1** if arr[k] has **(bit-i , bit-i+1) as (“11”)**

KEY DIFFERENCE 2 : SCAN ARRAYS

```
//Produce scan arrays
sum_scan_blelloch(d_scan_0, d_compact_0, n + 1);
sum_scan_blelloch(d_scan_1, d_compact_1, n + 1);
sum_scan_blelloch(d_scan_2, d_compact_2, n + 1);
sum_scan_blelloch(d_scan_3, d_compact_3, n + 1);
```

- Here, 4 scan arrays are produced using Blelloch Scan, for each 2-bit iteration
- 1 scan[] array for each compact[] array

EFFICIENCY OF 4-WAY RADIX SORT

- Finally, by using the 4 scan arrays as scatter address arrays, we can produce arr[] array rearranged according to bit-(i, i+1) split.
- Note that in 4-way radix sort, we **can arrive at sorted output (for integers) only after 16 iterations** (1 iteration for 1 pair of bits).
  - As compared to 2-way radix sort, where we need to do 32 iterations to arrive at sorted output
- Hence**, 4-way radix sort is more efficient** than normal 2-way radix sort

Step Complexity and Work Complexity

- **Step Complexity: O(1)**
  - **Totally 16 steps (One Step for 1 Bit)**
  - **Each step, n work is done parallelly**
- **Work Complexity : O(n)**