**Lab Exercise 1.3– Simple if-else in CUDA Program**

**Objective**

* Learn how to use if-else logic inside a CUDA kernel.
* Understand how different threads can make decisions independently.

**Program:**

#include <iostream>

#include <cuda\_runtime.h>

\_\_global\_\_ void checkEvenOdd(int \*a, int \*b, int N) {

int tid = blockIdx.x \* blockDim.x + threadIdx.x; // Global thread ID

if (tid < N) {

if (a[tid] % 2 == 0) {

b[tid] = 0; // Even number

} else {

b[tid] = 1; // Odd number

}

}

}

int main() {

const int N = 6;

int h\_a[N] = {1, 2, 3, 4, 5, 6}; // Input array

int h\_b[N]; // Output array

int \*d\_a, \*d\_b;

cudaMalloc(&d\_a, N \* sizeof(int));

cudaMalloc(&d\_b, N \* sizeof(int));

cudaMemcpy(d\_a, h\_a, N \* sizeof(int), cudaMemcpyHostToDevice);

checkEvenOdd<<<1, N>>>(d\_a, d\_b, N);

cudaMemcpy(h\_b, d\_b, N \* sizeof(int), cudaMemcpyDeviceToHost);

std::cout << "Result (0 = Even, 1 = Odd):\n";

for (int i = 0; i < N; i++) {

std::cout << h\_a[i] << ": " << h\_b[i] << "\n";

}

cudaFree(d\_a);

cudaFree(d\_b);

return 0;

}

**Step-by-Step Execution and Explanation**

**Step 1: Include Headers**

#include <iostream>

#include <cuda\_runtime.h>

* iostream: For standard input/output (std::cout).
* cuda\_runtime.h: Provides CUDA API functions.

**Step 2: Define the CUDA Kernel**

\_\_global\_\_ void checkEvenOdd(int \*a, int \*b, int N)

* \_\_global\_\_: Kernel function (runs on GPU, called from CPU).
* Purpose: Check if each element is **even** or **odd** and store the result:
  + 0 for even numbers.
  + 1 for odd numbers.

**Step 3: Calculate Global Thread ID**

int tid = blockIdx.x \* blockDim.x + threadIdx.x;

* tid identifies the **thread's global index** (position in the array).

**Step 4: Apply if-else Logic**

if (tid < N) {

if (a[tid] % 2 == 0) {

b[tid] = 0;

} else {

b[tid] = 1;

}

}

* Each thread checks:
  + If the element at index tid is divisible by 2 (% 2 == 0).
    - If **yes**, it’s **even** → store 0 in output array b.
    - Else, it’s **odd** → store 1 in output array b.

**Step 5: Host Code: Main Function**

const int N = 6;

int h\_a[N] = {1, 2, 3, 4, 5, 6};

int h\_b[N];

* Define two arrays:
  + h\_a: Input array {1, 2, 3, 4, 5, 6}.
  + h\_b: Output array (empty initially).

**Step 6: Allocate Device Memory**

cudaMalloc(&d\_a, N \* sizeof(int));

cudaMalloc(&d\_b, N \* sizeof(int));

* Allocate memory on the **GPU** for input (d\_a) and output (d\_b) arrays.

**Step 7: Copy Input Data to Device**

cudaMemcpy(d\_a, h\_a, N \* sizeof(int), cudaMemcpyHostToDevice);

* Transfer input array h\_a to device memory d\_a.

**Step 8: Launch Kernel**

checkEvenOdd<<<1, N>>>(d\_a, d\_b, N);

* 1 block with N=6 threads.
* Each thread handles one array element.

**Step 9: Copy Output Data from Device to Host**

cudaMemcpy(h\_b, d\_b, N \* sizeof(int), cudaMemcpyDeviceToHost);

* Transfer result array d\_b back to host array h\_b.

**Step 10: Print Results**

std::cout << "Result (0 = Even, 1 = Odd):\n";

for (int i = 0; i < N; i++) {

std::cout << h\_a[i] << ": " << h\_b[i] << "\n";

}

* Output the original value and its corresponding even/odd result.

**Step 11: Free Device Memory**

cudaFree(d\_a);

cudaFree(d\_b);

* Release the GPU memory.

**Expected Output**

Result (0 = Even, 1 = Odd):

1: 1

2: 0

3: 1

4: 0

5: 1

6: 0

* 1 → odd → 1
* 2 → even → 0
* 3 → odd → 1
* 4 → even → 0
* 5 → odd → 1
* 6 → even → 0

**Summary**

* **if-else** inside a CUDA kernel allows **threads to make decisions** independently.
* Each thread **processes one element** and decides what output to store.
* CUDA programming can handle **branching** just like normal C++ programming.