# CUDA Fibonacci Sequence on Google Colab: Debugging and Resolution Documentation

## Objective

To compute the Fibonacci sequence up to N = 220 using a CUDA kernel in Google Colab, compare its output with a sequential CPU implementation, and resolve all execution issues encountered due to the Colab environment.

## User Prompts & Results Summary

### Prompt 1: “Challenge #14: Fibonacci sequence in CUDA”

**Requested:** CUDA kernel for computing Fibonacci sequence for N = 220; compare with sequential CPU implementation.

### Prompt 2: “Guide me through implementing CUDA using Google Colab and give the solution for the above question”

**Requested:** Full environment setup and working solution in Google Colab. **Result:** CUDA kernel written, but produced all zeros.

### Prompt 3: “Mismatch at index 1: CPU 1 GPU 0 … F(1) = 0 … F(19) = 0”

**Diagnosis:** Kernel output returned prematurely; required cudaDeviceSynchronize().

### Prompt 4: “Give me the fixed code”

**Action Taken:** Provided full working .cu file with kernel computing Fibonacci and cudaDeviceSynchronize(). **Result:** Still mismatch; values all zero.

### Prompt 5: “Still not utilizing GPU properly”

**Diagnosis:** Threads running independently without dependency checks. **Fix Attempt:** Changed to single-thread kernel.

### Prompt 6: “Mismatch at index 2: CPU 1 GPU 0 …”

**Diagnosis:** Device memory not initialized or not accessible correctly inside the kernel.

### Prompt 7: “Give me the fixed code”

**Action Taken:** Added full kernel rewrite with manual device-side initialization and single-thread control logic. **Result:** Still failed due to PTX error.

### Prompt 8: “GPUassert: the provided PTX was compiled with an unsupported toolchain”

**Diagnosis:** PTX mismatch due to default nvcc arch in Colab.

### Prompt 9: “This is my version of GPU in Google Colab” (screenshot shows T4 selected)

**Resolution:** Confirmed Colab uses Tesla T4 (Compute Capability 7.5).

### Prompt 10: “It worked”

**Fix Implemented:** Used !nvcc -arch=sm\_75 fibonacci.cu -o fibonacci **Outcome:** Program compiled and executed correctly.

## Final Working Solution

### Fix Attempt 5: Specify Correct GPU Architecture

* Used:

!nvcc -arch=sm\_75 fibonacci.cu -o fibonacci

* Matched with Tesla T4 GPU (Compute Capability 7.5)

### Result:

* ✅ GPU output matched CPU.
* ✅ No mismatches.
* ✅ Correct Fibonacci values printed.

## Final Code Summary (Kernel + Error Checking + Synchronization)

\_\_global\_\_ void fibonacci\_kernel(unsigned long long\* fib, int N) {  
 if (threadIdx.x == 0 && blockIdx.x == 0) {  
 fib[0] = 0;  
 if (N > 1) fib[1] = 1;  
 for (int i = 2; i < N; i++) {  
 fib[i] = fib[i-1] + fib[i-2];  
 }  
 }  
}  
  
#define gpuErrchk(ans) { gpuAssert((ans), \_\_FILE\_\_, \_\_LINE\_\_); }  
inline void gpuAssert(cudaError\_t code, const char \*file, int line, bool abort=true) {  
 if (code != cudaSuccess) {  
 fprintf(stderr,"GPUassert: %s %s %d\n", cudaGetErrorString(code), file, line);  
 if (abort) exit(code);  
 }  
}

### Compilation Command:

!nvcc -arch=sm\_75 fibonacci.cu -o fibonacci

## Learnings and Key Takeaways

* CUDA kernels are asynchronous — always use cudaDeviceSynchronize().
* Device memory must be explicitly initialized if you’re depending on known values.
* Threads in CUDA cannot depend on values from other threads unless explicitly synchronized — Fibonacci must be computed sequentially.
* Google Colab often has version mismatches; always use -arch=sm\_75 for Tesla T4.
* Add error checking macros in CUDA C++ for debugging (cudaGetErrorString).

## Final Outcome:

✅ CUDA kernel working in Google Colab with Tesla T4 GPU.  
✅ Fibonacci sequence matched between CPU and GPU for N = 220.  
✅ All debugging steps and user prompts documented thoroughly with full error chain and resolutions.