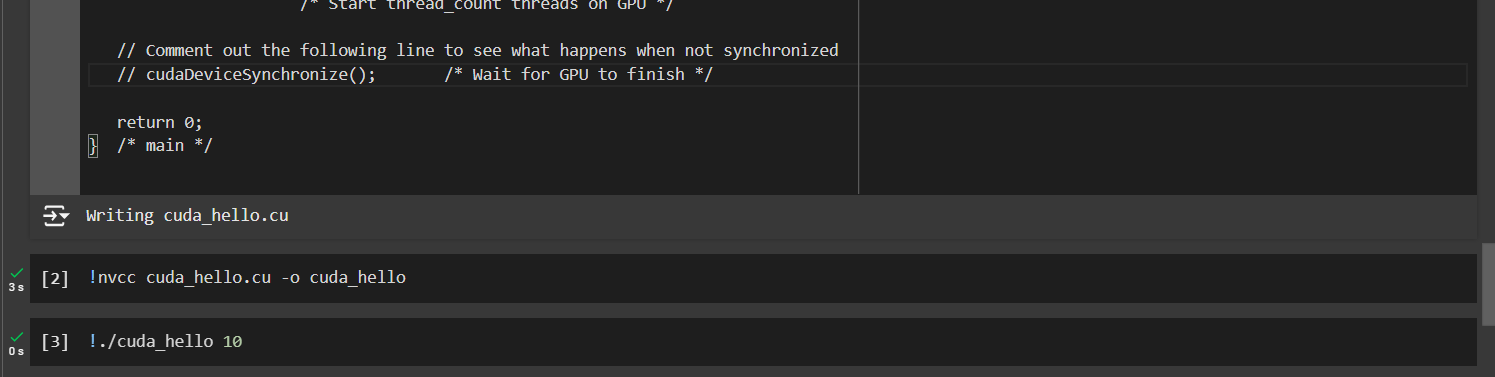
**6.1 Run the cuda\_hello program (Program 6.1) without the call to cudaDeviceSynchronize. What happens?**

****

I commented the cudaDeviceSynchronize call. Compiled and ran the program. However, there was no output. This is because the function ensures that the CPU waits for the GPU to finish executing all preceding tasks, without the call the main function may complete its execution and terminate the program before the GPU has a chance to complete the kernel execution.

**6.2 When we ran cuda\_hello (Program 6.1) with 10 threads, the output was ordered by the threads’ ranks: the output from thread 0 was first, then the output from thread 1, then the output from thread 2, etc.**

**Is this always the case?**

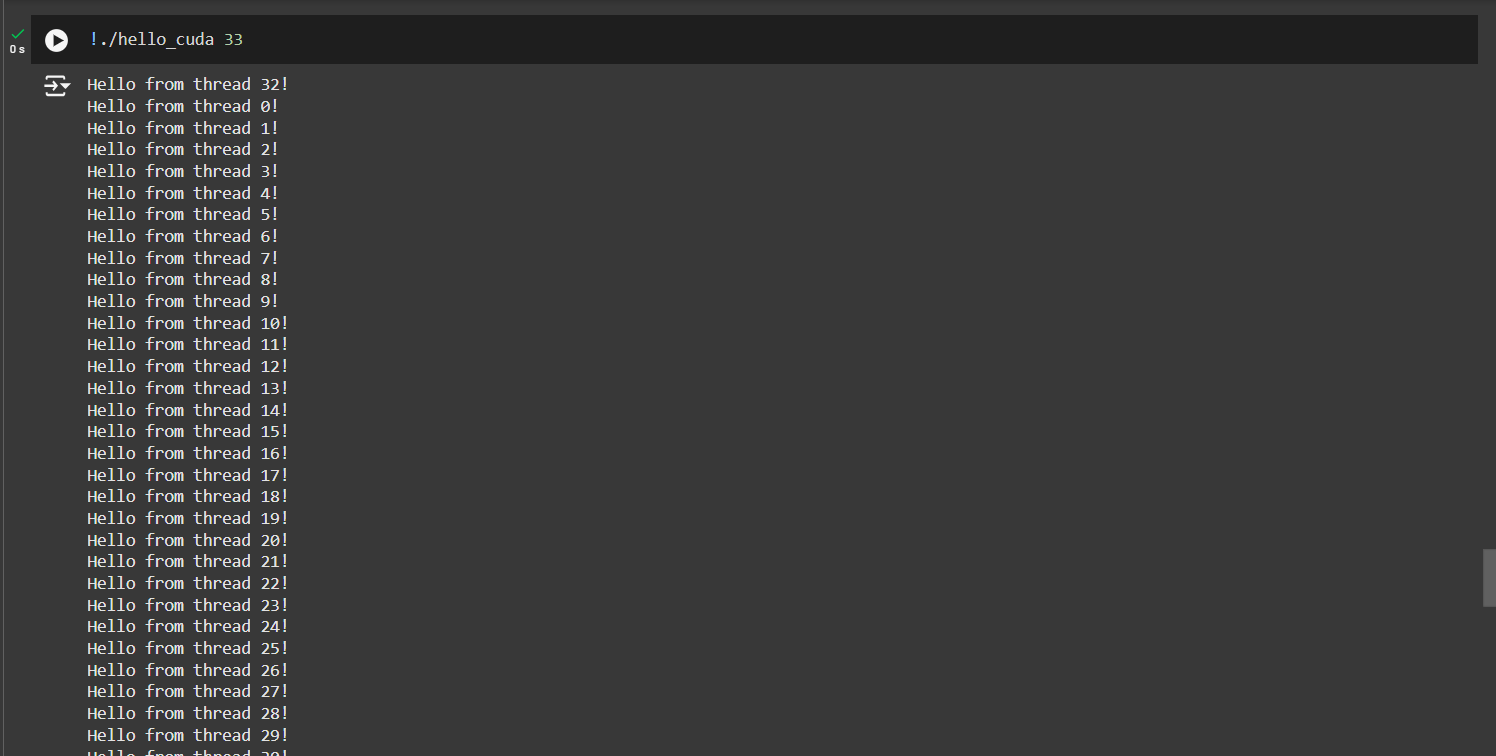
No, the output is not always ordered by the threads’ ranks.

**If not, what is the smallest number of threads you can start for which the output is not always ordered by the threads’ ranks?**

Typically, 33 or more threads, as this exceeds the size of a single warp (32 threads).

**Can you explain why?**

The GPU schedules warps independently and asynchronously, which can lead to non-deterministic output when multiple warps are involved.



At 33 threads the order changes, and may vary every time it’s run. It does not run it in order from 0 to 32.

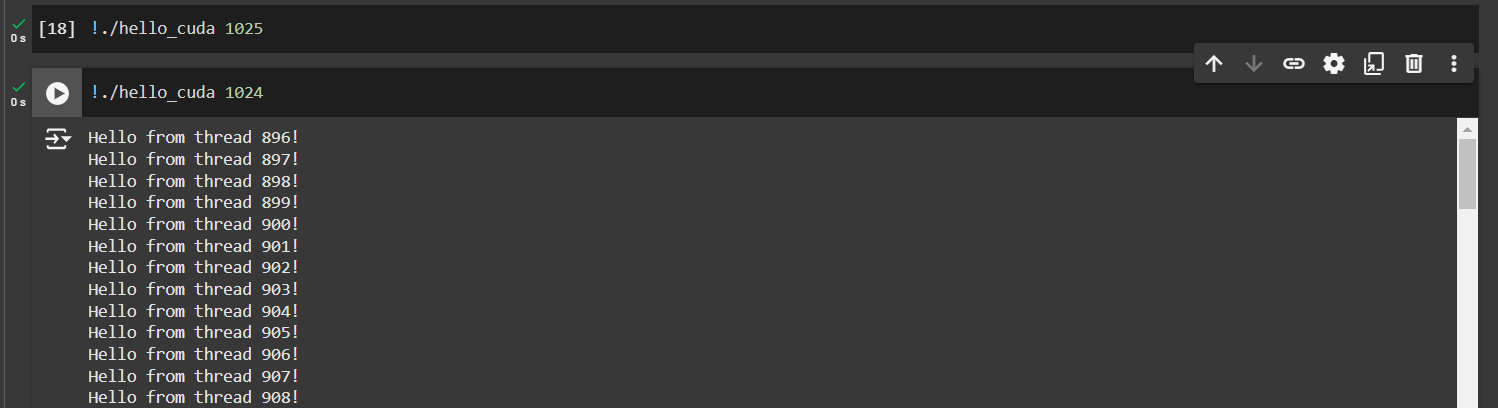
**6.3 When you run the cuda\_hello program (Program 6.1),**

**What is the largest number of threads you can start?**

1024 threads per block, but it is possible to launch multiple blocks.

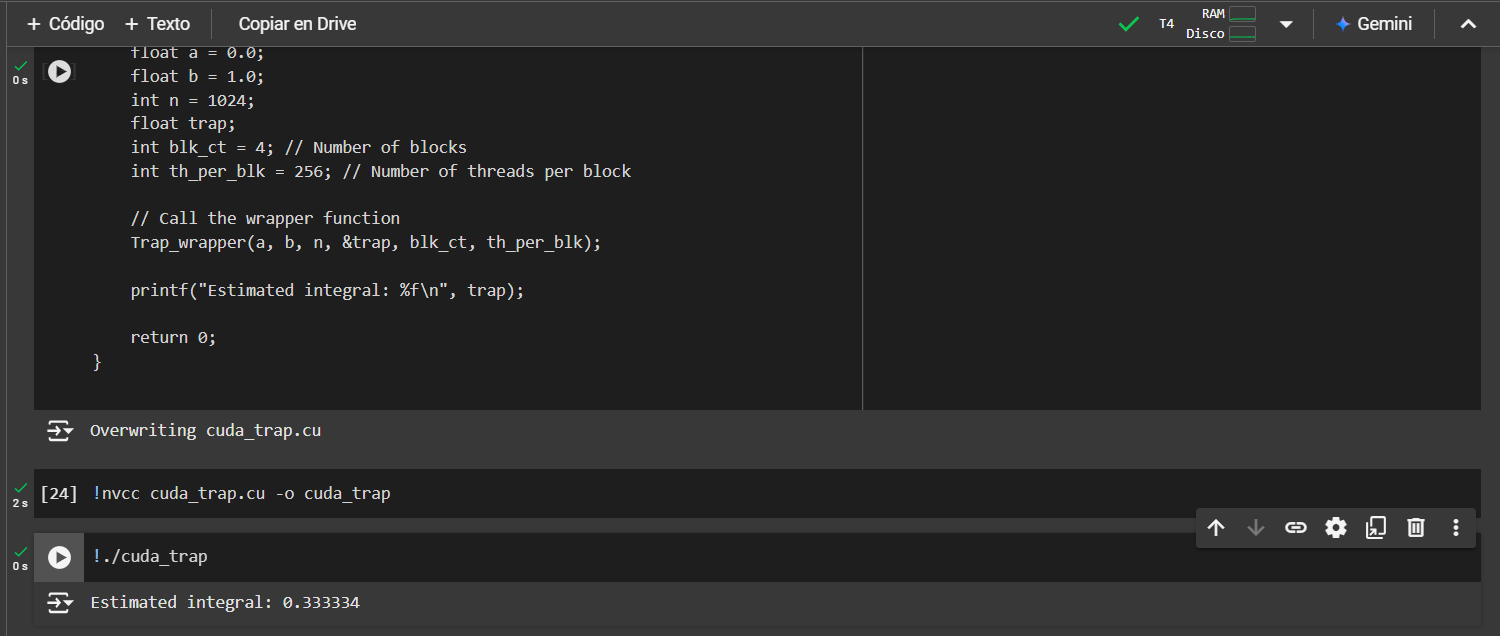
**What happens if you exceed this number?**

In google colabs, when I run with 1025 threads it simply does not run the program. I do not get any error. However, if I do it with 1024 it does run it without problems.



**6.4 Modify our first implementation of the trapezoidal rule (Program 6.12) so that it can be used on a system that doesn’t support unified memory.**

Code attached separately in a zip file.

****

**6.5** **Use the \_\_shfl\_sync function to implement a broadcast across a warp, i.e.,** **every thread in the warp gets a value stored in a designated thread of the warp.** **Write a driver kernel that calls your broadcast function, and a driver main** **program that initializes data structures, calls the kernel, and prints the results.**

Code attached separately in a zip file.

