Lab #06

Accelerating Applications using Multiple Cores

Assignment Type:	Individual
Assignment Duration:	1 week
Marks (of course total):	3%
Submission Method:	https://tcd.blackboard.com
Submission Deadline:	23:59 on Friday the 18th of March 2022

Introduction

This lab will introduce you to the concept of accelerating code execution using multiple CPU cores to run an application in parallel. The code will need to be run on the real hardware as the Wokwi simulator does not currently support multi-core code execution.

You should take the code you wrote for lab #02 (the Wallis approximation algorithm for calculating PI) and use that as the starting point for this lab. Your code should use one function that calculates the Wallis product using single-precision floating-point and a second function that calculates the Wallis product using double-precision floating-point.

Both functions should take the number of iterations of the algorithm to use as an argument and you should calculate the runtime of each algorithm (using the RP2040 built in timer hardware) as well as the total execution time of both algorithms when run on a single CPU core.

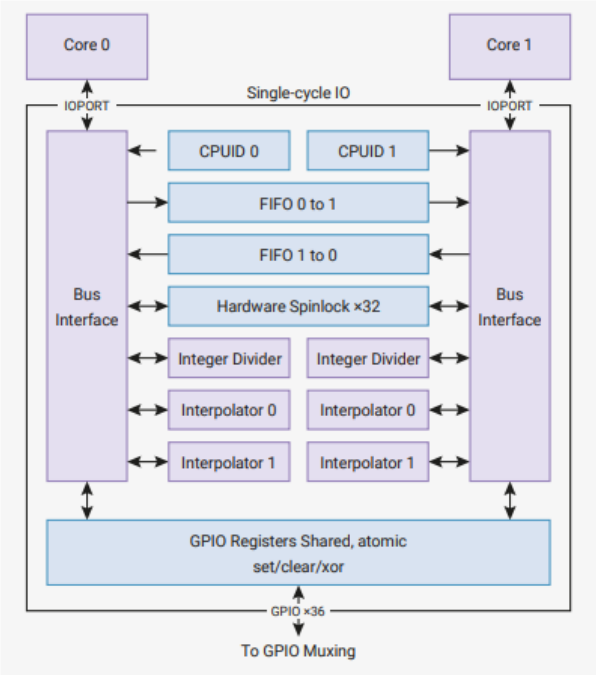

Figure 1: RP2040 internal components that allows the CPU cores to communicate with one another.

Finally, you should extend your application so that each of the functions runs on a separate CPU core in parallel and capture the total execution time to compare with the previous (sequential) version.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "pico/stdlib.h"
#include "pico/float.h"     // Required for using single-precision variables.
#include "pico/double.h"    // Required for using double-precision variables.
#include "pico/multicore.h" // Required for using multiple cores on the RP2040.

/**
 * @brief This function acts as the main entry-point for core #1.
 *        A function pointer is passed in via the FIFO with one
 *        incoming int32_t used as a parameter. The function will
 *        provide an int32_t return value by pushing it back on
 *        the FIFO, which also indicates that the result is ready.
 */
void core1_entry() {
    while (1) {
        //
        int32_t (*func)() = (int32_t(*)()) multicore_fifo_pop_blocking();
        int32_t p = multicore_fifo_pop_blocking();
        int32_t result = (*func)(p);
        multicore_fifo_push_blocking(result);
    }
}

// Main code entry point for core0.
int main() {

    const int    ITER_MAX   = 100000;

    stdio_init_all();
    multicore_launch_core1(core1_entry);

    // Code for sequential run goes here…
    //    Take snapshot of timer and store
    //    Run the single-precision Wallis approximation
    //    Run the double-precision Wallis approximation
    //    Take snapshot of timer and store
    //    Display time taken for application to run in sequential mode

    // Code for parallel run goes here…
    //    Take snapshot of timer and store
    //    Run the single-precision Wallis approximation on one core
    //    Run the double-precision Wallis approximation on the other core
    //    Take snapshot of timer and store
    //    Display time taken for application to run in parallel mode

    return 0;
}
```

After completing this lab, you should have successfully created and compiled a C application to run and display the results from the Wallace product algorithm for single-precision and double-precision values when running sequentially using a single CPU core and then adapted this code so that it will run one of the algorithms in parallel on the second CPU core.

Instructions

1. Make sure that your "pico-apps" repository is up to date before starting the lab.
2. Copy the code you wrote from "labs/lab02/lab02.c" into "labs/lab06/lab06.c" in the *"labs/lab06"* template folder.
3. Find a suitable timer function in the Pi Pico SDK to help you calculate and display:
    - The individual runtime for the single-precision approximation function
    - The individual runtime for the dual-precision approximation function
    - The total runtime for the application when running on a single CPU core
4. Using the example code from "*examples/multi_c*", adapt your code so that it runs the same tests as above except with one of the approximation functions running on the second CPU core in parallel instead of sequentially,
    - hat the total runtime should reflect running on both CPU cores in parallel.
5. Use the timer function from the Pi Pico SDK to help you calculate and display:
    - The individual runtime for the single-precision approximation function
    - The individual runtime for the dual-precision approximation function
    - The total runtime for the application when running using two CPU cores.
6. In a new text file called "lab06_rslt", add the following information:
    - The single-precision function execution time when using a single-core
    - The double-precision function execution time when using a single-core
    - Total execution time for running the single-precision and double-precision functions in sequentially using a single CPU core
    - The single-precision function execution time when using both cores
    - The double-precision function execution time when using both cores
    - Total execution time for running the single-precision and double-precision functions in parallel across both CPU core
    - Briefly comment on and discuss any interesting observations you make based on the various execution times you have logged
7. Make sure that the code is fully commented (Doxygen format is preferred)
8. Commit the changes you have made to your "*pico-apps"* main repository

Grading Scheme

Complete all steps in the instructions section and then upload the completed "**lab06.elf**", "**lab06.uf2**", "**lab06.c**" and the "**lab06_rslt.txt**" files to the lab06 assignment in Blackboard before the specified deadline to complete the lab. This lab is worth a total of 3% of your year-end results for the CSU232021 module and the marking scheme is as follows.

	INCOMPLETE [0%]	COMPETENT [50%]	PROFICIENT [100%]
SUBMISSION [10%]	Student has failed to submit the lab exercise.	Student has submitted the lab exercise but some of the required deliverables are missing.	Student has submitted the lab exercise and has included all the required deliverables.
IMPLEMENTATION [65%]	Student has not made a sufficient attempt at completing the lab exercise.	Student has completed the lab exercise however the implementation does not work as expected.	Student has completed the lab exercise and the implementation works as expected.
QUALITY [25%]	The code is badly structured or has not been well commented. Some or all of the specified tasks were not completed.	The code is adequately structured and there are some comments present. Some of the specified tasks were not completed.	The code is well structured with good and clear commenting throughout. All of the specified tasks were completed.