Assignment #1

Parallel and Distributed Computing

(All Sections)

Due date: 6th Feb, 2023, 10:00 PM

Instructions:

- Submit zip folder containing C files
- Plagiarism policy strictly applies to all the submissions
- Students with suspicious submissions will be called for viva
- Submit executable codes, else submissions will be discarded without evaluating them
- 30% deduction will be applied to assignments submitted at 11 PM. Submissions afterwards (i.e., 11:01 PM) will be awarded Zero.

Note:

To ensure that each thread is computed on a separate core/processor, we use CPU affinity or taskset. Binding or unbinding a process or a thread to a specific core or CPU inside the application is ensured through CPU affinity. However, taskset also can help you to achieve the same goal. To attempt this assignment, you need to have knowledge of the above concepts, the reference links are provided below.

https://linux.die.net/man/2/sched_getaffinity

https://man7.org/linux/man-pages/man1/taskset.1.html

Task 1:

**Use multithreading to attempt this task*

Write a program that performs the following operations:

- Execute the following commands to display information about your CPU. A list of computer specifications will appear with the information required to implement this question.
  - cat /proc/cpuinfo
  - cat /proc/meminfo
  - lscpu

Its up to you if you want to extract the information about the total no.of cores through above instructions(reading terminal's data and passing it as argument to the program) or use ***/proc/cpuinfo*** command inside your code.

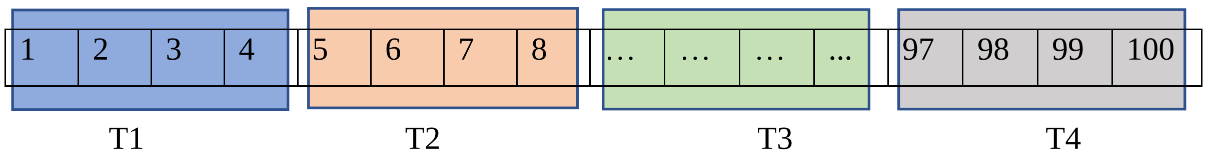
- Write a function to populate a 1D array of size 2^16.
- Write another function to calculate square root of each item of the array, also replacing it with the original item inside the array.

Implementation:

Now since you know the no.of cores alongside rest of the information regarding your CPU, implement the following task.

- Create no.of threads equal to no.of cores using ***pthread affinity*** or ***taskset.***
- Do block wise distribution of threads for **populate()** and **calcSqrt()** function, i.e., Assign the array items block wise to each thread in each function.
- Print the resultant array

Let's say you have 4 cores, create 1 thread per core and allocate array items to each thread as done in the following representation.



Constraints:

- Use the pthread_create() function to create threads.
- Use the pthread_join() function to wait for threads to complete.
- Use the pthread_setaffinity_np() function to assign each thread to a different processor core. While creating no.of threads, consider no.of logical cores as well.
- Calculate ***Variance of gflops*** by executing your program with affinity and without affinity.

Grading Criteria:

- Correct use of threads to perform the operations.
- Correct use of pthread_create(), pthread_join() and pthread_setaffinity_np().
- Correct computation and storage of the square root of all the items in the array.
- Correct printing of the square root of all the items in the array by the parent process (main function).
- Correct calculation of ***Variance of gflops*** by executing your program with affinity and without affinity.

Task 2:

**Use multithreading to attempt this task*

Write a program that performs the following operations:

- **Populate( )** a 2D array of size ***2^16*2^16*** with random integers (range 8000-16000).
- Compute the **sum( )** of all the elements in the array and store it in a global variable.
- Print the sum of all the elements in the array.

Implementation:

- *Version 1 (Block wise):* The data-computation or thread distribution should be block wise. Assign each bock to 1 thread. Divide the 2D array as per following block sizes.
  - ***2^2,***
  - ***2^4,***
  - ***2^8.***
- *Version 2 (Cyclic):* the data-computation by threads should be cyclic. no of threads should be equal to 2^2, each thread should execute fix no of rows, i.e., Each thread will be assigned 2^4 rows.
- Compute GFlops for each version and each of the block size.
- ***Draw plots for 3+1 scenarios (V1 and V2).***
- ***Draw block size/no of rows on X-axis, and gflops on Y-axis.***
- Print all the computed values on the screen with detailed description.

Constraints:

- Use the pthread_create() function to create threads.
- Use the pthread_join() function to wait for threads to complete.

Grading Criteria:

- Correct use of threads to perform the operations.
- Correct use of pthread_create(), pthread_join().
- Correct computation and storage of the sum of all the elements in the array.
- Correct computation of GFlops for each block.
- Correct plot representation for various thread execution schemes.
- Self-explanatory representation of all the computed values in order to deliver the clear meaning of each operation performed.
- Your submissions will be matched with online available codes. Plagiarism policy will be applied to all the copied submissions.