**COMSATS UNIVERSITY ISLAMABAD,**

**ABBOTTABAD CAMPUS.**

**COURSE TITTLE**: PARALLEL AND DISTRIBUTED PROGRAMMING

**FINAL PAPER**: PDC LAB.

**GROUP MEMBERS**: ASSAD U LLAH, UMER BASHIR.

**REGISTRATION NO**: FA18-BCS- 042, FA18-BCS-069.

**SUBMITTED TO**: SIR MATEEN YAQOOB.

**DATE:** **26TH JULY 2022**

* **LAB FINAL PROJECT:**
* **PLATFORM USED:**

*The platform we used for this computation was OpenMP because it is more compatible with our personal machine and also takes less space and we also had a prior experience of working on it so it makes it easier to understand approach for us.*

* ***DETAILED ANALYSIS:***

*The under observation code has two regions which are performing intensive tasks, one is Julia set function nested loops and the other is Julia function loops. In the Julia set function we can use the parallel approach and are parallelizing the nested loops because we want multiple threads to work at the multiple numbers of pixels for Julia computation so we can run the most compute intensive task in this code using parallelization approach of Open MP . In the Julia function we have data dependencies involved so we cannot parallelize that region of code.*

* ***screenshot of code with parallelization included:***

Text, letter

Description automatically generated

* **screenshot of code which cannot be parallelized:**

A picture containing scatter chart

Description automatically generated

* ***PARALLELIZATION STRATEGY:***

*Nested loops inside Julia set function were converted into parallel because it was performing high compute task where each pixel in the given set was checked for Julia point so we use multiple threads in it to check from e.g. 16 threads to work at the 16 pixels at the same time and find their Julia point and then it there is a Julia point exists then it will color it red and otherwise it will give it a white color and all this task was performed with the Julia function which was finding the point on the given coordinates now when it was working in parallel it was computing multiple pixels at one time so it was increasing its performance and we used pragma omp parallel and pragma omp for using this approach.*

* ***EXECUTION TIME AND SPEEDUP RESULTS:***

|  |  |  |
| --- | --- | --- |
| **THREADS** | **EXECUTION TIME (seconds)** | **SPEEDUP** |
| 1 | 0.186 | 1.60215054 |
| 2 | 0.298 | 1 |
| 3 | 0.242 | 1.23140496 |
| 4 | 0.165 | 1.80606061 |
| 5 | 0.188 | 1.58510638 |
| 6 | 0.169 | 1.76331361 |
| 7 | 0.17 | 1.75294118 |
| 8 | 0.149 | 2 |
| 9 | 0.162 | 1.83950617 |
| 10 | 0.144 | 2.06944444 |
| 11 | 0.157 | 1.89808917 |
| 12 | 0.136 | 2.19117647 |
| 13 | 0.138 | 2.15942029 |
| 14 | 0.13 | 2.29230769 |
| 15 | 0.149 | 2 |
| 16 | 0.129 | 2.31007752 |
|  |  |  |
|  |  |  |

* ***RESULTS:***

*When we run this code in the serial way or using one thread applying on the region that is performing compute intensive task then its execution time is equal to 0.298 seconds but when we use 2 threads to compute the part which is compute intensive then the execution time drops to 0.18 which is a significant change in value, although the execution time between results fluctuates when we are applying multiple threads but it always remains greater than the execution time which was calculated on the serial values.*