|  |  |  |  |
| --- | --- | --- | --- |
| **Solution Type** | **Number of Slaves** | **Execution time** | **Explain the result** |
| Sequential Solution | 1 | 8.680009 | This result is of the ‘regular’ sequential execution, as MPI was not used in this solution we get the straightforward time it took to execute ‘Heavy’ |
| Static Task Pool | 2 | 5.937294 | Two process executed a predetermined portion of the tasks (50% each) and then reduce was called to sum the result of each task. There is an improvement to the ‘sequential’ execution. in this solution, the final sum might be delayed in case one of the processes is slower from the other. Execution time will be limited by the time the slowest process takes to execute. |
| Static Task Pool | 4 | 4.618956 | 4 processes ran a predetermined portion of the tasks (25% each). this is why we see an improvement in execution time. |
| Dynamic Task Pool | 2 | 9.028125 | 2 processes used to execute Heavy, in the way It was implemented, one process is established as the master and one as the Slave. when using 2 processes there is not much sense in it, the master will be in charge of a single process that will execute all tasks. this is why the single slave in sequential execution is better time, we save the time it takes to create the master. |
| Dynamic Task Pool | 4 | 3.032981 | A great improvement of the execution time, where each slave executes a very small task and gets a new one upon completion. This is why this is better than the 4 slaves in static execution, there might be a slave that is faster when comparing to other slaves, and his quick execution time can be utilized as opposed to the 4 tasks in static where each gets a predetermined amount of tasks. |
| Dynamic Task Pool | 20 | 1.277951 | Here we can see the true power of dynamic task pool execution where a single master oversees 20 slaves where is executes a single tasks and gets a new one upon completion, allows the fastest slave to execute more tasks. |