CSCE 735 – Parallel Computing HW 3 – Sidharth Edupuganti

1:

Text

Description automatically generated

These results showcase that the code is working as expected and is reporting error=0.

2:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **K = 12** | **q** | **threads** | **time** | **speedup** | **efficiency** |
| 12 | 0 | 1 | 0.0069 | 1 | 1 |
| 12 | 1 | 2 | 0.0059 | 1.169492 | 0.584746 |
| 12 | 2 | 4 | 0.0063 | 1.095238 | 0.27381 |
| 12 | 4 | 16 | 0.015 | 0.46 | 0.02875 |
| 12 | 6 | 64 | 0.0343 | 0.201166 | 0.003143 |
| 12 | 8 | 256 | 0.2981 | 0.023147 | 9.04E-05 |
| 12 | 10 | 1024 | 3.8213 | 0.001806 | 1.76E-06 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **K = 20** | **q** | **threads** | **time** | **speedup** | **efficiency** |
| 20 | 0 | 1 | 0.1851 | 1 | 1 |
| 20 | 1 | 2 | 0.1003 | 1.845464 | 0.922732 |
| 20 | 2 | 4 | 0.0591 | 3.13198 | 0.782995 |
| 20 | 4 | 16 | 0.0298 | 6.211409 | 0.388213 |
| 20 | 6 | 64 | 0.0515 | 3.594175 | 0.056159 |
| 20 | 8 | 256 | 0.1239 | 1.493947 | 0.005836 |
| 20 | 10 | 1024 | 4.0279 | 0.045954 | 4.49E-05 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **K = 28** | **q** | **threads** | **time** | **speedup** | **efficiency** |
| 28 | 0 | 1 | 62.9315 | 1 | 1 |
| 28 | 1 | 2 | 32.0775 | 1.961858 | 0.980929 |
| 28 | 2 | 4 | 16.3897 | 3.839698 | 0.959925 |
| 28 | 4 | 16 | 4.8041 | 13.09954 | 0.818721 |
| 28 | 6 | 64 | 3.8781 | 16.22741 | 0.253553 |
| 28 | 8 | 256 | 5.3666 | 11.72651 | 0.045807 |
| 28 | 10 | 1024 | 8.6057 | 7.312769 | 0.007141 |

The results of this experiment do align with the typical behavior of parallel programming. The whole idea of parallel programming is that multiple tasks are running concurrently to cut down on runtime of a process. This type of behavior is being displayed here within the graphs. As larger list sizes increase (, the there is an increase in speed up, especially when q = 6. However, after it reaches this point the speedup decreases. This may be due to the number of threads that have increased; however the efficiency of each thread decreases as q increases. Therefore, the efficiency decreases despite there being an increase in speedup. The threads are being underutilized. Due to the increase in threads, they are not all able to stay efficient. This also results in some overhead, hence the extra time, which is shown from the data tables. These graphs do align with the principles of parallel programming. These results are very similar to the outputs to the previous assignment, which is another indication that the results are correct because OpenMP allows for a much easier to way to be able to create and joining threads by creating parallel regions, which is better than creating and joining multiple threads.

3: Execution Time with different Affinity and Places

|  |  |  |  |
| --- | --- | --- | --- |
|  | **threads** | **cores** | **sockets** |
| **master** | 67.6812 | 67.4923 | 5.1998 |
| **close** | 3.1174 | 3.1167 | 3.116 |
| **spread** | 3.1273 | 3.0906 | 3.1071 |

There are a plethora of observations that can be made from these results. We can see that when the OMP\_PROC\_BIND = spread, the 3 abstract names are relatively same. With OMP\_PROC\_BIND = master, the results are vastly different, where when OMP\_PLACES = threads, and cores, the runtime is roughly 67, as seen from the table above. This is because when the bind is set to master, all threads are assigned to the same place as the master thread, which causes more overhead. When OMP\_PLACES = sockets, the runtime decreases greatly under master. This is due to sockets being composed of cores, which is composed of threads. The sockets are a culmination of both threads and cores, so despite being placed in the same area as the master thread, the runtime will be lower than threads, or cores. For all other values, the runtime is relatively low, around 3 seconds, despite the binding being set to close or spread.