**CSYE7105 HW2**

Shivani Shekhawat

002656554

Time-Elapsed Observations for Apply and Map Functions with Different CPU Numbers

|  |  |  |
| --- | --- | --- |
| Number of Cpus | Elapsed Time(map) | Elapsed Time(apply) |
| 1 | 0.51 seconds | 9.45 seconds |
| 2 | 0.51 seconds | 8.33 seconds |
| 4 | 0.51 seconds | 8.71 seconds |
| 8 | 0.49 seconds | 10.39 seconds |

A graph with a line going up

Description automatically generated

Elapsed Time vs Number of CPUs:

* **apply() and map() Elapsed Time Comparison**

The first plot compares the elapsed time for two different functions, apply() and map(), as the number of CPUs increases.

elapsed\_time\_apply\_list and elapsed\_time\_map\_list store the elapsed time for each function at different CPU counts.

* **Trend Expectations**

Generally, as the number of CPUs increases, parallelizable tasks tend to show decreased elapsed time due to parallel processing.

Both lines (apply() and map()) are expected to decrease as the number of CPUs increases, but the rate of decrease might vary depending on the nature of the tasks being parallelized and the efficiency of parallelization.

* **Parallelization Efficiency**

If the tasks are well-suited for parallelization and there is minimal overhead in managing parallel processes, you should see a noticeable reduction in elapsed time as the number of CPUs increases.

* Depending on the nature of the tasks and the system's capabilities, there might be a point where increasing the number of CPUs does not lead to a significant reduction in elapsed time. This is because of diminishing returns or overhead associated with managing parallel processes.

A graph of a line

Description automatically generated with medium confidence

Speedup vs Number of CPUs

* **Speedup calculation**

Speedup is calculated as the ratio of the elapsed time with a single CPU to the elapsed time with a certain number of CPUs.

speedup\_apply and speedup\_map store the speedup values for each function at different CPU counts.

* **Trend Expectation**

Speedup values greater than 1 indicate an improvement compared to the single CPU case.

The speedup should ideally increase with the number of CPUs, but the rate of increase may slow down due to factors like diminishing returns and overhead.

* **Parallelization Efficiency Impact**

If the tasks are highly parallelizable and the parallelization is efficient, we see a significant increase in speedup as the number of CPUs increases.

Similar to the elapsed time plot, there is a point where the speedup increases at a slower rate. This could be due to the factors mentioned earlier.

So,

As the number of CPUs went from less to more, the elapsed time also increased. That is, more CPUs do not achieve the expected speedup.

Assumption 1: The number of iterations is only 200, which is very small, so the error will be large.

Assumption 2: As more CPU cores are used, each CPU core slows down as more are used, also generates more overhead.

Assumption 3: Contention for cache memory and contention for main memory—CPU cores are forced to wait on main memory (far slower than registers).

A blue rectangles with white lines

Description automatically generated

We are using chunk sizes of (300, 200), (1000, 1000), and (3000, 2000). The smaller chunk size of (300, 200) may lead to higher overhead due to the increased number of tasks that Dask needs to manage. The larger chunk sizes of (1000, 1000) and (3000, 2000) may reduce this overhead, but they could also lead to higher memory usage. The optimal chunk size will depend on the specific characteristics of your data and computations, as well as the available memory.

**(300, 200) Chunk Size:**

This chunk size is the smallest among the tested sizes. Smaller chunks increase the overhead associated with parallelizing the computation and can result in longer execution times. While smaller chunks can be beneficial for some tasks, they may not be the most efficient choice for all operations.

**(1000, 1000) Chunk Size:**

This chunk size is relatively larger, and Dask can efficiently process larger chunks, as they allow for better parallelism and reduced overhead when processing data. Therefore, the computation time is relatively faster compared to smaller chunks.

**(3000, 2000) Chunk Size:**

This chunk size is smaller than the original chunk size but still relatively large. It offers some level of parallelism but doesn't have the same advantage as the original chunk size. However, it's still faster than the smallest chunk size.