**CSYE7105 13969 Parallel Machine Learning & AI SEC 01 Fall 2024**

Submitted by: Anshul Chaudhary

NEU ID: 002820096

Professor: Handan Liu

**Part 2: Normalization**

**Q5.1 Describe the information (hostname, CPU and memory) of the node(s) you use for calculation on Discovery. [1 point]**

**A:**

Details are mentioned on the table below:

|  |  |
| --- | --- |
| **Host** | c0747 |
| **CPU** | 8 cores |
| **Memory** | 8 GB |

**Q5.2 Create a table and fill in the wall-clock time (seconds or minutes) you obtained with the different CPU numbers in Q2, Q3 and Q4. [3 points]**

**A:**

The table below has all the details about wall clock time:

|  |  |  |
| --- | --- | --- |
| **Method** | **CPU** | **Time (Sec)** |
| map | 2 | 0.5761 |
| 4 | 0.4056 |
| 6 | 0.4729 |
| 8 | 0.3921 |
| imap  Chunk\_size 10 | 2 | 0.4746 |
| 4 | 0.3817 |
| 6 | 0.4274 |
| 8 | 0.4193 |
| imap  Chunk\_size 50 | 2 | 0.4239 |
| 4 | 0.4773 |
| 6 | 0.7124 |
| 8 | 0.4869 |

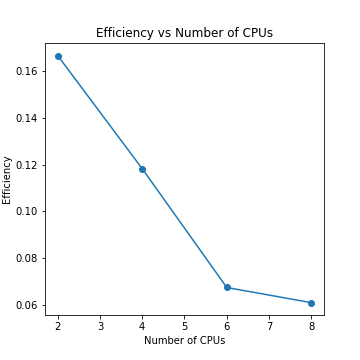
**Q5.3 Insert the graph images of Q2, Q3 and Q4 appropriately in this Word document [3pt] and analyze your results: the speedup, efficiency and overhead; and why you think one certain result is the optimal result. [4pt]**

**A:**

1. Parallel computing using pool.map

**A graph with numbers and lines

Description automatically generatedA blue line on a white background

Description automatically generated**

Based on the graph above we can observe that:

* The wall-clock time drops sharply from 2 to 4 CPUs, but it increases slightly beyond 4. And decreases again at 8 CPUs.
* The speedup initially increases but then decreases drastically to 6 CPUs before improving slightly at 8 CPUs. This behavior suggests non-linear scaling.
* The dip in speedup at 6 CPUs could indicate that task management overhead became substantial at that point. Another possibility is that the workload wasn’t distributed evenly across all CPUs.
* As the number of CPUs increases, efficiency drops significantly.
* This behavior is expected as adding more processors can lead to diminishing returns in terms of parallel performance. Overhead increases with more CPUs, and the workload might not be evenly distributed, resulting in lower efficiency.

1. Parallel performance using pool.imap

A graph with blue lines and dots

Description automatically generatedA blue line with dots

Description automatically generatedA blue line with dots on a white background

Description automatically generated

**A blue line with dots

Description automatically generatedA graph with blue dots and a black background

Description automatically generatedA blue line with dots on a white background

Description automatically generated**

Based on the graph above we can observe that:

* From the wall clock time in chunks 10 and 50 we can observe that the time increases as the cores increase in chunk size 50, whereas for size 10 there was a sudden decrease and then a slight increase with an increase in cores.
* The performance of 10 chunk size is better than that of 50 as we can see from speed up as well, the speedup decreases in chunk size 50 whereas there is an increase in speedup in chunk size 10.
* As the number of CPUs increases, efficiency drops significantly in both cases.
* The dip in speedup after 4 CPUs could indicate that task management overhead became substantial at that point in both cases.

1. Compare pool.map vs pool.imap(chunk size 10)

A graph of a line and a triangle

Description automatically generated with medium confidence

* The wall-clock time shows a mixed pattern, particularly with 4 and 6 CPUs, where it increases slightly before dropping again with 8 CPUs.
* The chunk size 10 scenario shows a smoother wall-clock time drop but still has slight fluctuations, likely indicating that some configurations lead to better load balancing than others.
* At 4 CPUs we can see that imap with chunk size as 10 is taking significantly less time.

A screenshot of a graph

Description automatically generated

* For both the normal case and the chunk size 10 case, the speedup does not increase linearly and shows variability. For the non-chunked case, speedup increases significantly from 2 to 4 CPUs but drops at 6 CPUs before increasing again at 8 CPUs. A similar pattern is observed for the chunk size 10 case, with fluctuations around CPU counts.
* The irregular speedup may indicate overhead from synchronization, communication, or a workload imbalance among CPUs, especially at higher counts.

A graph of a line

Description automatically generated with medium confidence

* Efficiency decreases sharply with more CPUs in both cases (with and without chunk size 10). This is typical when overheads grow as the number of processors increases.
* The efficiency starts at a maximum with 2 CPUs but decreases significantly with 8 CPUs, meaning the work distribution among CPUs becomes less effective at higher counts.

**Overhead:** The overhead includes synchronization, communication, and data transfer times, which become more significant as the number of CPUs increases. The speedup and efficiency curves, combined with irregularities in wall-clock time, indicate that there is considerable overhead that hinders performance.

**Optimal result:** Based on these plots, the best performance (minimal wall-clock time with relatively better speedup) seems to occur around 4 CPUs. While efficiency drops significantly, the overall speedup and wall-clock time suggest that 4 CPUs achieve the fastest execution time despite the overhead. This implies that chunking helps mitigate some overhead, making 4 CPUs more optimal than lower configurations.

**Part 2: Correlation**

**Q5.1 Describe the information (hostname, CPU and memory) of the node(s) you use for calculation on Discovery. [1 point]**

**A:**

Details are mentioned on the table below:

|  |  |
| --- | --- |
| **Host** | c0747 |
| **CPU** | 8 cores |
| **Memory** | 8 GB |

**Q5.2 Create a table and fill in the wall-clock time (seconds or minutes) you obtained with the different CPU numbers in Q2, Q3 and Q4. [3 points]**

**A:**

The table below has all the details about wall clock time:

|  |  |  |
| --- | --- | --- |
| **Method** | **CPU** | **Time (Sec)** |
| map | 2 | 7.3981 |
| 4 | 5.7648 |
| 6 | 5.2899 |
| 8 | 5.0853 |
| imap  Chunk\_size 10 | 2 | 6.5398 |
| 4 | 4.7461 |
| 6 | 4.6100 |
| 8 | 4.6020 |
| imap  Chunk\_size 50 | 2 | 6.1929 |
| 4 | 4.8498 |
| 6 | 4.3973 |
| 8 | 4.5176 |

**Q5.3 Insert the graph images of Q2, Q3 and Q4 appropriately in this Word document [3pt] and analyze your results: the speedup, efficiency and overhead; and why you think one certain result is the optimal result. [4pt]**

1. Parallel computing using pool.map

**A blue line on a white background

Description automatically generatedA blue line with dots

Description automatically generatedA blue line on a white background

Description automatically generated**

Based on the graph above we can observe that:

* The speedup shows improvement with more CPUs, but efficiency drops because the additional CPUs do not contribute proportionally to reducing the execution time. This happens due to communication overhead, which grows as more processors need to communicate and synchronize their work.
* While adding more CPUs reduces wall-clock time, the optimal number of CPUs appears to be around 4 or 6, where the efficiency is still reasonable, and the speedup is noticeable.
* The returns on adding CPUs diminish due to overhead, as seen from the decreasing efficiency and marginal speedup gains between 6 and 8 CPUs. While adding more CPUs reduces wall-clock time, the optimal number of CPUs appears to be around 4 or 6.

1. Parallel performance using pool.imap

A blue line with dots on a white background

Description automatically generatedA blue line on a white background

Description automatically generatedA blue line with dots on a white background

Description automatically generated

A blue line with dots

Description automatically generatedA blue line with a black background

Description automatically generatedA blue line on a white background

Description automatically generated

Based on the graph above we can observe that:

* From the wall clock time in chunks 10 and 50 we can observe that the time decreases as the cores increase, in chunk size 50 we can observe that the wall clock time is less.
* The performance of 50 chunk size is better than that of 10 as we can see from speed up as well, the speedup is low in chunk size 10 whereas there is an increase in speedup at chunk size 50.
* As the number of CPUs increases, efficiency drops significantly in both cases.
* The dip in speedup after 6 CPUs could indicate that task management overhead became substantial at that point in both cases.

1. Compare pool.map vs pool.imap(chunk size 50)

A graph of a graph of a graph

Description automatically generated with medium confidence

* Wall-clock time decreases as the number of CPUs increases, with chunk size 50 achieving lower times overall. However, after 6 CPUs, there’s minimal improvement and even a slight increase at 8 CPUs with chunking.
* Wall-clock time measures the actual time taken for the execution. More CPUs reduce the time required for completion, but due to communication overhead and load balancing inefficiencies, adding too many CPUs can cause performance degradation, especially if the workload per CPU becomes too small.
* The chunk size helps minimize this overhead, allowing more efficient CPU usage up to a certain point (around 6 CPUs), after which the overhead outweighs the parallelization benefits.

A graph of a line

Description automatically generated with medium confidence

* Speedup improves as the number of CPUs increases, but there are diminishing returns, especially in the non-chunked case. With chunk size 50, the speedup reaches a higher value (about 1.35) compared to without chunking (~1.15)
* Speedup is a measure of how much faster a parallel process runs compared to a serial one. The ideal scenario is linear speedup, where doubling the number of CPUs results in a halving of runtime. However, overhead and coordination costs between CPUs reduce the actual speedup.
* The chunking strategy (with size 50) allows for better parallelization by reducing the communication and coordination costs, leading to a higher speedup. This effect is especially noticeable when going from 2 to 4 CPUs, after which the benefits plateau.

A graph of a line and a point

Description automatically generated with medium confidence

* Efficiency declines as the number of CPUs increases in both cases (with and without chunk size 50). The rate of decline is slightly slower with chunk size 50.
* Efficiency measures how well computational resources (CPUs) are being utilized. Ideally, it should stay constant as you increase the number of CPUs, but due to overhead and communication costs, efficiency tends to drop when more CPUs are added.
* The chunk size of 50 slightly mitigates this effect because tasks are batched, reducing the communication overhead between CPUs. However, there is still a diminishing return as the number of CPUs increases.

**Optimal Result:** The optimal result seems to occur with 6 CPUs and chunk size 50, where speedup is maximized, and wall-clock time is minimized. This is likely because at this point, the balance between parallel computation and overhead is most favorable. Adding more CPUs beyond this point leads to diminished returns due to the increasing overhead from coordination and communication between CPUs. The chunk size of 50 allows for more efficient batching, but at 8 CPUs, the workload per CPU might be too small, increasing the overhead.