Q5. Create a Word document named **hw2.docx**. In this Word document, you should do:         [11 points]

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

ANS ->

Hostname -> c0639

Cpu Config ->

CPUs – 28 Cores

Model – Intel(R) Xeon(R) CPU E5-2680 v4 @ 2.4GHz

Memory Config ->

Memory Block Size -> 2GB

Total Online Memory -> 256 GB

PART 2

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]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Number of Cores | Map Values | imap values  (chunk size = 3) | imap values  (chunk size = 7) |
| Wall – Clock (s) | 2 | 0.166 | 0.287 | 0.331 |
|  | 4 | 0.143 | 0.203 | 0.288 |
|  | 6 | 0.145 | 0.199 | 0.268 |
|  | 8 | 0.157 | 0.211 | 0.258 |
|  |  |  |  |  |
| Speed up | 2 | 1.528 | 1.127 | 0.948 |
|  | 4 | 1.777 | 1.564 | 1.090 |
|  | 6 | 1.750 | 1.625 | 1.170 |
|  | 8 | 1.618 | 1.531 | 1.217 |
|  |  |  |  |  |
| Efficiency | 2 | 0.764 | 0.563 | 0.474 |
|  | 4 | 0.444 | 0.398 | 0.272 |
|  | 6 | 0.291 | 0.270 | 0.195 |
|  | 8 | 0.202 | 0.191 | 0.152 |

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] 7 points].

For Part 2 Q2, of DF-normalization of pool.map implementation:

A graph with a line graph and numbers

AI-generated content may be incorrect.

A graph with a line and a number of cpus

AI-generated content may be incorrect.A graph with a green line

AI-generated content may be incorrect.

*Let’s look at the first graph of Wall-clock time vs CPUs:*

The wall clock time seems to decrease till we reach number of cpus added to 4,

*Speedup:*

Even the max speed up seems to appear at when n-cpus = 4, after that it starts to trail off, with efficiency keeps on dropping as we add more cores showing realistic parallel computing phenomenon experience that doubling on processor doesn’t double the performance.

*Efficiency and Overhead:* it seems to take time and slow down suggesting cpu overhead caused by synchronization and concatenation, creation and management of new worker threads on initialization of pool . After examining the graph its safe to assume optimum performance is achieved when number of cpus = 4.

For Part 2 Q3, of pool.imap implementation:

A graph with a line and a number of cpus

AI-generated content may be incorrect.A graph with a line and a number of cpus

AI-generated content may be incorrect.A graph of efficiency and number of cpus

AI-generated content may be incorrect.

*Wall-Clock Time and Speedup:*

Looking the graph, we can clearly see imap when used at chunksize=3 performs better when chunksize = 7 in every aspect i.e wall clock time, speed up and efficiency

Best performance is achieved when CPUs=6 when chuncksize=3, it runs fastest and achieves best speedup, although the efficiency seems to drop as we add more cores. But

*Efficiency:*

It’s much more efficient compared to chunksize=7 at 6 cores, while performance is good at 4 cores, 6 cores seem little more optimum.

*Overhead:*

In case when cores=6 and chunksize=3, We do get little overhead by spending more efficiency to achieve little performance boost. We can argue whether we want to sacrifice efficiency for that little bit of speedup and decrease in wall clock time or not. But considering we want to get done as fast as possible core=6 seems good. As we hit serialization after core 6. And larger chunk size is equal to a larger overhead in this case.

For Part2, Q.4 map v imap analysis:A graph with a line and a line graph

AI-generated content may be incorrect.A graph with blue and orange lines

AI-generated content may be incorrect.A graph with blue and orange lines

AI-generated content may be incorrect.

*In terms of Wall-clock time ->* The map method works better than faster than imap for every core we added, at core=4, map method seems to be optimum as the time slightly increases when core=6.

*Speed up:*

we can map method beats imap method. Even here the highest speed up is achieved at when core=4. It plateaus at core=6 and worsens core=8 and will worsen as we increase cores. This happens because cores spend more time coordinating than computing also known as synchronization overhead.

*Efficiency and Overhead:*

As for Efficiency, it keeps on decreases as we add more cores due to above mentioned overhead, but imap method’s efficiency seems to be better than map since imap scales linearly at early steps, it also streams result as soon as each chunk finished and even usage of CPUs.

**Part 3**

Q5. Create a Word document named **hw2.docx**. In this Word document, you should do:         [11 points]

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

ANS ->

Hostname -> c0639

Cpu Config ->

CPUs – 28 Cores

Model – Intel(R) Xeon(R) CPU E5-2680 v4 @ 2.4GHz

Memory Config ->

Memory Block Size -> 2GB

Total Online Memory -> 256 GB

PART 3

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]

For Part 3 Q5.2, of DF-correlation computation

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Number of Cores | Map Values | imap values  (chunk size = 3) | imap values  (chunk size = 7) |
| Wall – Clock (s) | 2 | 0.231 | 0.293 | 0.337 |
|  | 4 | 0.196 | 0.199 | 0.292 |
|  | 6 | 0.187 | 0.212 | 0.265 |
|  | 8 | 0.203 | 0.218 | 0.261 |
|  |  |  |  |  |
| Speed up | 2 | 1.405 | 1.102 | 1.010 |
|  | 4 | 1.658 | 1.626 | 1.163 |
|  | 6 | 1.738 | 1.525 | 1.283 |
|  | 8 | 1.600 | 1.483 | 1.304 |
|  |  |  |  |  |
| Efficiency | 2 | 0.702 | 0.551 | 0.505 |
|  | 4 | 0.414 | 0.406 | 0.290 |
|  | 6 | 0.289 | 0.254 | 0.213 |
|  | 8 | 0.200 | 0.185 | 0.163 |

For Part 3 Q2, of DF-correlation with pool.map implementation: A graph with a line

AI-generated content may be incorrect.A graph with a line and a line

AI-generated content may be incorrect.A graph with a green line

AI-generated content may be incorrect.

*Analysis:*

*Let’s look at the first graph of Wall-clock time vs CPUs and Speedup:*

The wall clock time seems to decrease till we reach number of cpus added to 6.

*Efficiency:*

Even the max speed up seems to appear at when n-cpus = 6, after that it starts to trail off, with efficiency keeps on dropping as we add more cores showing realistic parallel computing phenomenon experience that doubling on processor doesn’t double the performance.

*Overhead:*

it seems to take time and slow down suggesting cpu overhead caused by synchronization and concatenation, creation and management of new worker threads on initialization of pool . After examining the graph its safe to assume optimum performance is achieved when number of cpus = 6.

For Part 3 Q3, of DF-correlation with pool.imap implementation:

A graph with blue and orange lines

AI-generated content may be incorrect.A graph with blue and orange lines

AI-generated content may be incorrect.A graph with blue and orange lines

AI-generated content may be incorrect.

Looking the graph, we can clearly see imap when used at chunksize=3 performs better when chunksize = 7 in every aspect i.e wall clock time, speed up and efficiency

*Speed up:*

Best performance is achieved when CPUs=4 when chuncksize=3, it runs fastest and achieves peak speedup. If we increase more cores the time increases, speedup and efficiency decreases.

*Efficiency:*

Although the efficiency seems to drop as we add more cores for chunksize = 3. But it’s much more efficient compared to chunksize=7 at 2, 4, 6, 8 cores.

*Overhead:*

Chunk size of 3 seems to spread evenly and keep each CPU busier, in multiprocessing if we keep chunk size too small it generates excessive scheduling overhead and if we keep it too big (like chunk size = 7), there is more CPU wait time. Chunk size=3 seems to balance that overhead compared to size=7.

For Part 3 Q4, of DF-correlation with map vs imap: (next page)

A graph with blue and orange lines

AI-generated content may be incorrect.A graph with blue and orange lines

AI-generated content may be incorrect.A graph with blue and orange lines

AI-generated content may be incorrect.

*Wall-Clock and Speed Up:*

Map is consistently faster than imap across all CPU Cores. At 4-6 CPUs the map hits the lowest(0.196 – 0.187 seconds) while imap at (0.199 – 0.212 seconds) making imap slightly slower than map.

Even for speedup map peaks at 1.738 for 6-cores, while imap peaks at 1.626 for 4 cores. In this case map at 4 core core(1.658) is still slightly better than imap. Although both tend to saturate after 4-6 CPU cores which is a clear sign of Amdahl’s law kicking in.

*Efficiency:*

This is where it gets little nuanced, the imap curve is above of map curve for all core counts. Suggesting imap although slow in time, works more efficiently by utilizing CPU resources evenly . since it dynamically assigns new chunks, it leads to better load balancing. The task uniformity also changed in this case.

*Overhead:*

Beyond 4-6 CPUs, both speedup and efficiency curves flatten and rise, signalling synchronization overhead, serialization and final concatenation.