Ulysses Palomar

Wednesday, November 23, 2022

Parallel Programming Comparison of a Multithreaded Program

The purpose of this assignment is to compare the efficiency of threads by comparing run times for a program running single threaded or multithreaded. This program will populate a grid with values dependent on the surrounding cells. At the end, the program will print statistics about its run.

To accomplish this, I used a java program I created which is able to run single threaded or multithreaded, depending on the user’s preference. The program will start at the beginning of the grid, sum its surrounding cells’ values, and calculate the average. That new average will be saved to that middle cell, and the program will move on to the next cell. The program will also calculate the error for that cell by subtracting the previous average of that cell from the new average. After running through the whole grid, all the errors will be summed up. The program will run until the total error is less than 5, otherwise, it’ll make another iteration. I ran this on my personal laptop which has a reported 14 cores and 20 logical processors (see image below).

Graphical user interface, application

Description automatically generated with medium confidence

In theory, the more threads I used, the faster the program should run. So, the run time for 4 threads should theoretically be ¼ of the time 1 thread took. However, using multiple threads can cause variances in correctness. The final grid value for the middle cell should be 50. But, when multiple threads get added into the mix, the final calculations can be different each time the program is ran. This is because when 2 threads are working on cells or rows that touch each other, the first thread might calculate the average before the other second thread was able to save its average to the cell that the first thread will use in its calculations. For my data, I will be printing the starting grid (border has default values and center cells are 0), the errors as it runs (to make sure the program is running correctly), and the final grid. The first case I tested was calculating the grid with a single thread. The second test was calculating the grid using 4 unsynchronized threads. The third test was calculating the grid using 4synchronized threads. The fourth test was testing 4 synchronized threads while SSH’ed into the lab computers. Here are my results:

For reference, this is a screenshot of my CPU usage while no code is running:

Chart

Description automatically generated

This is a screenshot of my CPU usage while testing Single Thread - Grid Size = 500 w/ Printing Error and my data:

Chart, histogram

Description automatically generated

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Single Thread - Grid Size = 500 w/ Printing Error | | | | | | |
| Trial | 1 | 2 | 3 | 4 | 5 | Average |
| Iterations | 105257 | 105257 | 105257 | 105257 | 105257 | 105257 |
| Average | 49.49139 | 49.49139 | 49.49139 | 49.49139 | 49.49139 | 49.49139 |
| Error | 4.99 | 4.99 | 4.99 | 4.99 | 4.99 | 4.99 |
| Time (ms) | 279000 | 276000 | 282000 | 283000 | 347000 | 293000 |
| Est. Time | 69750 | 69000 | 70500 | 70750 | 86750 | 73250 |
| ESpeedUp |  |  |  |  |  | 4 |

This is a screenshot of my CPU usage while testing Single Thread - Grid Size = 500 w/o Printing Error and my data:

Chart, histogram

Description automatically generated

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Single Thread - Grid Size = 500 w/o Printing Error | | | | | | |
| Trial | 1 | 2 | 3 | 4 | 5 | Average |
| Iterations | 105257 | 105257 | 105257 | 105257 | 105257 | 105257 |
| Average | 49.49139 | 49.49139 | 49.49139 | 49.49139 | 49.49139 | 49.49139 |
| Error | 4.99 | 4.99 | 4.99 | 4.99 | 4.99 | 4.99 |
| Time | 208000 | 186000 | 229000 | 229000 | 224000 | 215000 |
| Est. Time | 52000 | 46500 | 57250 | 57250 | 56000 | 53750 |
| ESpeedUp |  |  |  |  |  | 4 |

This is a screenshot of my CPU usage while testing Unsynchronized Multithread (4) - Grid Size = 500 w/ Printing Error and my data:

Graphical user interface, application, table

Description automatically generated

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Unsynchronized Multithread (4) - Grid Size = 500 w/ Printing Error | | | | | | |
| Trial | 1 | 2 | 3 | 4 | 5 | Average |
| Iterations | 183070 | 179514 | 209288 | 168001 | 161122 | 180199 |
| Average | 46.63212 | 44.52875 | 48.68812 | 48.67367 | 48.53579 | 47.41169 |
| Error | 4.99999 | 4.99999 | 4.99999 | 4.99999 | 4.99999 | 4.99999 |
| Time | 97900 | 97500 | 87500 | 96800 | 87900 | 93520 |
| ASpeedUp |  |  |  |  |  | 3.13302 |

This is a screenshot of my CPU usage while testing Unsynchronized Multithread (4) - Grid Size = 500 w/o Printing Error and my data:

Chart

Description automatically generated

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Unsynchronized Multithread (4) - Grid Size = 500 w/o Printing Error | | | | | | |
| Trial | 1 | 2 | 3 | 4 | 5 | Average |
| Iterations | 138989 | 163508 | 146040 | 116862 | 158308 | 144741.4 |
| Average | 41.76434 | 42.69981 | 41.56983 | 40.43458 | 41.91334 | 41.67638 |
| Error | 4.99999 | 4.99999 | 4.99999 | 4.99999 | 4.99999 | 4.99999 |
| Time | 88500 | 87200 | 90100 | 86400 | 85700 | 87580 |
| ASpeedUp |  |  |  |  |  | 2.454898 |

This is a screenshot of my CPU usage while testing Synchronized Multithread (4) - Grid Size = 500 w/ Printing Error and my data:

Chart, histogram

Description automatically generated with medium confidence

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Synchronized Multithread (4) - Grid Size = 500 w/ Printing Error | | | | | | |
| Trial | 1 | 2 | 3 | 4 | 5 | Average |
| Iterations | 105262 | 105272 | 105266 | 105271 | 105268 | 105267.8 |
| Average | 49.49116 | 49.49119 | 49.49111 | 49.49113 | 49.49113 | 49.49114 |
| Error | 4.99973 | 4.99937 | 4.9999 | 4.99983 | 4.99905 | 4.999576 |
| Time | 104500 | 110400 | 99530 | 108200 | 104040 | 105334 |
| ASpeedUp |  |  |  |  |  | 2.781628 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Synchronized Multithread (4) - Grid Size = 500 w/ Printing Error | | | | | | |
| Trial | 1 | 2 | 3 | 4 | 5 | Average |
| Iterations | 105262 | 105272 | 105266 | 105271 | 105268 | 105267.8 |
| Average | 49.49116 | 49.49119 | 49.49111 | 49.49113 | 49.49113 | 49.49114 |
| Error | 4.99973 | 4.99937 | 4.9999 | 4.99983 | 4.99905 | 4.999576 |
| Time | 104500 | 110400 | 99530 | 108200 | 104040 | 105334 |
| ASpeedUp |  |  |  |  |  | 2.781628 |

This is a screenshot of my CPU usage while testing Synchronized Multithread (4) - Grid Size = 500 w/o Printing Error and my data:

Chart, histogram

Description automatically generated

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Synchronized Multithread (4) - Grid Size = 500 w/o Printing Error | | | | | | |
| Trial | 1 | 2 | 3 | 4 | 5 | Average |
| Iterations | 105263 | 105263 | 105283 | 105265 | 105265 | 105267.8 |
| Average | 49.49112 | 49.49121 | 49.49129 | 49.49123 | 49.49117 | 49.4912 |
| Error | 4.99911 | 4.99947 | 4.99899 | 4.99935 | 4.99955 | 4.999294 |
| Time | 97850 | 96430 | 93570 | 95540 | 92340 | 95146 |
| ASpeedUp |  |  |  |  |  | 2.259685 |

This is a screenshot of my CPU usage while testing Synchronized Multithread (4) - Grid Size = 500 w/ Printing Error (SSH) and my data:

Graphical user interface, chart, histogram

Description automatically generated

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Error | 4.99973 | 4.99937 | 4.9999 | 4.99983 | 4.99905 | 4.999576 |
| Time | 104500 | 110400 | 99530 | 108200 | 104040 | 105334 |
| ASpeedUp |  |  |  |  |  | 2.781628 |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Synchronized Multithread (4) - Grid Size = 500 w/ Printing Error (SSH) | | | | | | |
| Trial | 1 | 2 | 3 | 4 | 5 | Average |
| Iterations | 105267 | 105264 | 105274 | 105276 | 105264 | 105269 |
| Average | 49.4912 | 49.49019 | 49.49115 | 49.4912 | 49.49099 | 49.49095 |
| Error | 4.99928 | 4.99953 | 4.99988 | 4.99932 | 4.99974 | 4.99955 |
| Time | 91050 | 90030 | 87030 | 89490 | 88480 | 89216 |
| ASpeedUp |  |  |  |  |  | 3.284164 |

This is a screenshot of my CPU usage while testing Synchronized Multithread (4) - Grid Size = 500 w/o Printing Error (SSH) and my data:

Chart, histogram

Description automatically generated

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Synchronized Multithread (4) - Grid Size = 500 w/o Printing Error (SSH) | | | | | | |
| Trial | 1 | 2 | 3 | 4 | 5 | Average |
| Iterations | 105258 | 105259 | 105257 | 105261 | 105260 | 105259 |
| Average | 49.49136 | 49.49137 | 49.49133 | 49.49138 | 49.49138 | 49.49136 |
| Error | 4.99985 | 4.99996 | 4.99864 | 4.99986 | 4.99984 | 4.99963 |
| Time | 91040 | 89820 | 90210 | 92060 | 91520 | 90930 |
| ASpeedUp |  |  |  |  |  | 2.364456 |

Below are graphs to help visualize the data above:

As you can see from the data, the program, for the most part, performed as expected. To start, the CPU utilization when no program was running, was low. Which makes sense since I had nothing else running in the background. The CPU utilization with a single thread showed a slight increase than no program. This is the one thread using up the CPU to run the program. The CPU utilization showed an even larger increase with unsynchronized and synchronized. This is from the 4 threads all using the CPU to make the calculations. But, when running the program via SSH, there was a significant decrease in CPU utilization. This is because the program is using the lab computer’s CPU, so my CPU does relatively no work. There are slight peaks in utilization, probably from the connection between my machine and the lab computer. My laptop still has to receive the results from the program running on the lab computer. Starting with the times graph, all multithread methods were significantly faster than the single thread method. It may not have been a perfect speedup, as you can see in the speed up graph, but it is definitely something worth appreciating. The imperfect speedup is expected in the sense that no program can be 100% efficient. It could also be my laptop’s power. When I ran the programs, it made sure to not do anything else on my machine while it ran. I didn’t want to mess with my results if I were to browse the web while running my program. Nonetheless, even with my attempts to conserve CPU utilization, a speedup of 4 was not achieved. All multithread methods performed similarly in regard tos time. But printing the total error during the whole process does increase the time. It is a slight increase in multithread, but the single thread method had a noticeable increase in time when printing the total error. This completely makes sense given that while the threads may have finished doing their work, the program still needs time to print the errors. Unsynchronized shows a slight decrease in time when implemented. This is because when the threads are synchronized, they must wait for each other. So, even if one thread works faster than another, it has to wait for all threads to catch up. This is what causes synchronized to have higher times than unsynchronized. While unsynchronized is faster than synchronized, it is not as accurate as synchronized. The average temperature value with the unsynchronized method was significantly lower (44.544035) than the synchronized methods. The synchronized methods, however, had a lower error compared to the unsynchronized and single thread methods. This can be attributed to the fact that when the threads are unsynchronized, one thread’s speed may affect another’s calculations. As I mentioned above, one thread might calculate the average before the other second thread was able to save its average to the cell that the first thread will use in its calculations. In any method, the error while continuously decrease as it runs. But with unsynchronized, the threads are not able to achieve a final average temperature as close to 50 as the other methods before the error goes below 5. Another thing to note is that the number of iterations in the unsynchronized method compared to the other methods. The single thread method will produce the same number of iterations with each run. This is normal since it’s one thread doing the same exact routine. It will produce the same results. And if you look at my data for the single thread method, it did produce the same exact results, for each category. However, when using multiple threads, since they can run at different speeds from one another, more iterations will be needed at times. This goes back to what I’ve mentioned before about a thread’s speed affecting another thread’s calculations. With the unsynchronized methods, it is more apparent that a thread can affect another thread’s calculations by the significant increase in the number of iterations. With the synchronized methods, there isn’t a significant increase, but the fact that there are variations in the number of iterations is proof that threads can affect threads.

Imagine having to mop the entire UW – Parkside campus by yourself. It is possible but will consume a lot of time. So, to reduce time, you enlist the help of 3 other friends. Together, you all finish much faster. This is the idea behind our single thread and multithread methods. In this homework, we were able to notice the difference between synchronized and unsynchronized threads. While multiple threads are indeed faster, if not kept in check, they can produce unwanted results. Whether you should prioritize speed, by using unsynchronized threads, or accuracy, by using synchronized threads, is entirely up to you. The differences in time between synchronized and unsynchronized is negligible so it makes most sense to prioritize accuracy, since you don’t lose a significant amount of time with synchronized compared to unsynchronized. If the time difference was much more noticeable, then the topic to prioritize is up to you. It would depend on whether accuracy is more crucial to your profession. For example, a lab testing cures for diseases would prioritize accuracy significantly more than time. But if you are using multiple threads in a program you made for fun, you might prioritize time over accuracy. It all depends. But for my program, with my results, I would prioritize accuracy because of what little amount of time you lose when doing so.