6G6Z0030 High Performance Computing & Big Data

William Husband
21359557

Declaration

No part of this project has been submitted in support of an application for any other degree or qualification at this or any other institute of learning. Apart from those parts of the project containing citations to the work of others, this project is my own unaided work. This work has been carried out in accordance with the Manchester Metropolitan University research ethics procedures, and has received ethical approval number Your EthOS Number.

Signed: 

High Performance Computing

Introduction

For the serial, OpenMP and MPI code then main optimisations utilised are the "-O0", "-O1", "-O2", "-O3" and "-fast" options that are provided by Intel (Intel Corporation, 2006). These all offer varying levels of optimisation. "-O0" disables any optimisation, "-O1" applies basic amounts of optimisation, "-O2" is the default optimisation level and "-O3" applies "aggressive" optimisation that is beneficial for applications which perform calculations, and "-fast" applies several common optimisations. The impact that all of these optimisations have on the performance of each of the serial, MPI and OpenMP code is explored, along with the impact that certain code alterations have in the case of MPI and OpenMP. The specific lines that are updated in OpenMP and MPI are in Appendix A.

Serial

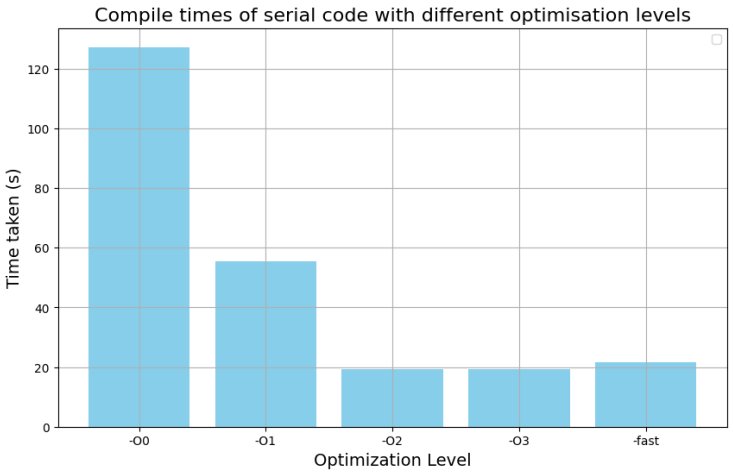
For the serial code, each .exe with a different Intel optimisation was run 3 times, then the average of these 3 was taken as the overall compile time. The "-O0" option resulted in the longest compile time of 127.262 seconds, the "-O1" optimisation resulted in the second longest compile time, and the other optimisations lead to similar low compile times, with "-O2" causing the lowest compile time of 19.36377 seconds. This means that the default optimisation option was the best performing option for optimising the serial code. The numerical results can be seen in Appendix A, which are visualised in a bar graph below. All of the runs of serial code gave the output of the centre of mass as (-0.09509,-0.16562,49.64602).

Code to make .exe:

```
icx -O0 md.c -o md_O0.exe
icx -O1 md.c -o md_O1.exe
icx -O2 md.c -o md_O2.exe
icx -O3 md.c -o md_O3.exe
icx -fast md.c -o md_fast.exe
```

Code to run .exe:

```
./md_O0.exe
./md_O1.exe
./md_O2.exe
./md_O3.exe
./md_fast.exe
```



OpenMP

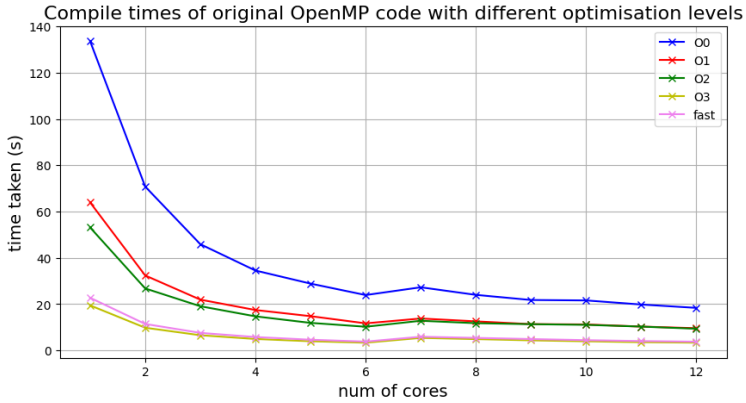
The original version of the OpenMP code for molecular dynamics was run with the Intel optimisations once using differing numbers of cores, from 1 core to 12 cores. This was done using a loop. No optimisation ("-O") once again led the the highest compilation times at every number of cores, however the best performing optimisations on the original OpenMP code were "-O3" and "-fast", with "-O3" performing slightly better, these results can be seen in Appendix A, the graph below visualises the compile times. The result for the centre of mass was (-0.09509,-0.16562,49.64602) for all runs. Every optimisation lead to an increased compile time when 7 cores are utilised, as seen in the graph below, leading to the possibility of a new overhead being introduced when 7 cores are utilised. This means that the code can likely be altered in future to mitigate this new overhead and improve parallel scalability.

Code to make .exe:

```
icx -O0 -qopenmp md-OpenMP.c -o md_OpenMP_O0.exe
icx -O1 -qopenmp md-OpenMP.c -o md_OpenMP_O1.exe
icx -O2 -qopenmp md-OpenMP.c -o md_OpenMP_O2.exe
icx -O3 -qopenmp md-OpenMP.c -o md_OpenMP_O3.exe
icx -fast -qopenmp md-OpenMP.c -o md_OpenMP_fast.exe
```

Code to run .exe:

```
for n in `seq 1 12`;do export OMP_NUM_THREADS=$n; ./md_OpenMP_O0.exe | tail -n 2; done
for n in `seq 1 12`;do export OMP_NUM_THREADS=$n; ./md_OpenMP_O1.exe | tail -n 2; done
for n in `seq 1 12`;do export OMP_NUM_THREADS=$n; ./md_OpenMP_O2.exe | tail -n 2; done
for n in `seq 1 12`;do export OMP_NUM_THREADS=$n; ./md_OpenMP_O3.exe | tail -n 2; done
for n in `seq 1 12`;do export OMP_NUM_THREADS=$n; ./md_OpenMP_fast.exe | tail -n 2; done
```



Updated OpenMP code

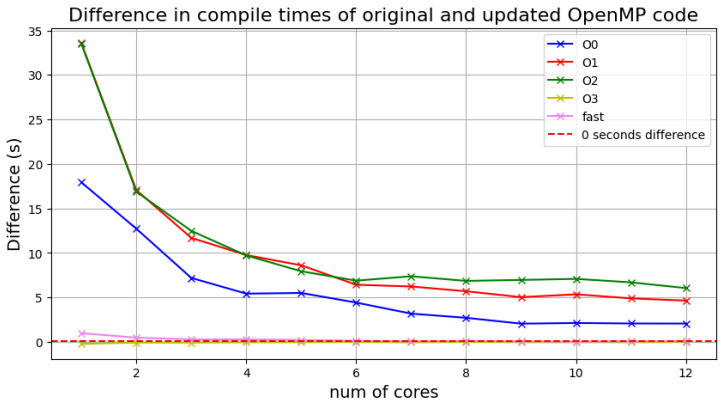
The original OpenMP code calculated values in a shared array, this was altered so to introduce local variables which are incremented within each thread in order to reduce part of the overhead, the new .c file is called “md-OpenMP_12.c” in the OneDrive folder. This was done using a “reduction” pragma statement. The local variables introduced were local versions of “vx”, “vy” and “vz”, called “vxLoca", “vyLocal” and “vzLocal”. The code updates can be seen in Appendix A. These allowed for more efficient calculations of the variables. This updated code was then made into .exe files with each of the Intel optimisations, each of which were run using numbers of cores from 1-12. The updated code had reduced compile times for almost all of the new .exe files, except “-O3” and “-fast” were relatively unaffected. The difference in all compile times can be seen numerically in Appendix A, the graph below shows the amount of time that the updated code reduces the compile time compared to the original code (original code compile time minus the new code compile time, so a large number means a large reduction in compile time). The results show that even using higher amounts of cores, the compile time is reduced with the updated code when running with most intel optimisation levels, showing that some level of overhead has been mitigated. The output of the centre of mass was again (-0.09509,-0.16562,49.64602) for all runs.

Code to make .exe:

```
icx -O0 -qopenmp md-OpenMP_12.c -o md_OpenMP_12_O0.exe
icx -O1 -qopenmp md-OpenMP_12.c -o md_OpenMP_12_O1.exe
icx -O2 -qopenmp md-OpenMP_12.c -o md_OpenMP_12_O2.exe
icx -O3 -qopenmp md-OpenMP_12.c -o md_OpenMP_12_O3.exe
icx -fast -qopenmp md-OpenMP_12.c -o md_OpenMP_12_fast.exe
```

Code to run .exe:

```
for n in `seq 1 12`;do export OMP_NUM_THREADS=$n; ./md_OpenMP_12_O0.exe | tail -n 2;
done
for n in `seq 1 12`;do export OMP_NUM_THREADS=$n; ./md_OpenMP_12_O1.exe | tail -n 2;
done
for n in `seq 1 12`;do export OMP_NUM_THREADS=$n; ./md_OpenMP_12_O2.exe | tail -n 2;
done
for n in `seq 1 12`;do export OMP_NUM_THREADS=$n; ./md_OpenMP_12_O3.exe | tail -n 2;
done
for n in `seq 1 12`;do export OMP_NUM_THREADS=$n; ./md_OpenMP_12_fast.exe | tail -n 2;
done
```



MPI

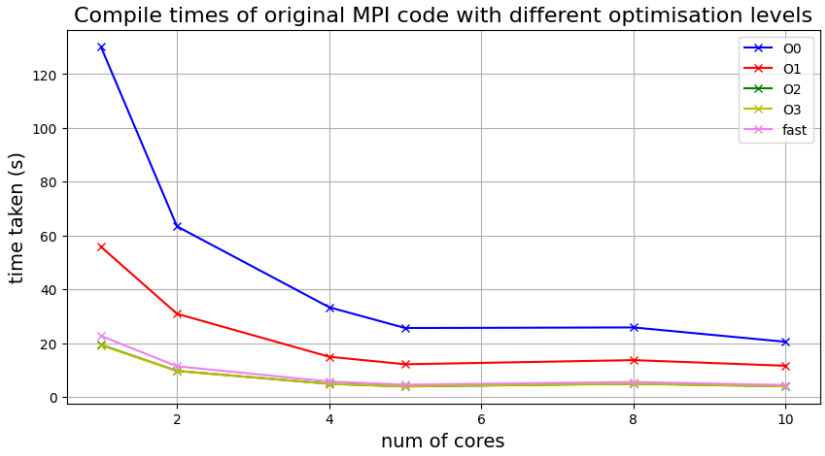
The original MPI code was also compiled as .exe files with each of the Intel optimisations, all of which were run on differing numbers of cores from 1-12. There was an issue with this code where it would abort if the number of cores was not a factor of 20000. However, on all the runs that did not abort, the result was (-0.09509,-0.16562,49.64602) for the centre of mass. The compile times were reduced after after increasing the number of cores, however a slight increase in compile time is introduced when 8 cores are utilised, which can be seen in the graph below. This once again leads to the potential of additional overheads being introduced with this number of cores, which would mean code alterations are necessary to improve parallel scalability. No optimisation "-O0" lead the the highest compile times, while "-O2" and "-O3" performed almost identically with the lowest compile times.

Code to make .exe:

```
mpiicx -O0 md-MPI.c -o md_MPI_O0.exe
mpiicx -O1 md-MPI.c -o md_MPI_O1.exe
mpiicx -O2 md-MPI.c -o md_MPI_O2.exe
mpiicx -O3 md-MPI.c -o md_MPI_O3.exe
mpiicx -fast md-MPI.c -o md_MPI_fast.exe
```

Code to run .exe:

```
for n in `seq 1 12`;do mpirun -np $n ./md_MPI_O0.exe | tail -n 2; done
for n in `seq 1 12`;do mpirun -np $n ./md_MPI_O1.exe | tail -n 2; done
for n in `seq 1 12`;do mpirun -np $n ./md_MPI_O2.exe | tail -n 2; done
for n in `seq 1 12`;do mpirun -np $n ./md_MPI_O3.exe | tail -n 2; done
for n in `seq 1 12`;do mpirun -np $n ./md_MPI_fast.exe | tail -n 2; done
```



Updated MPI code

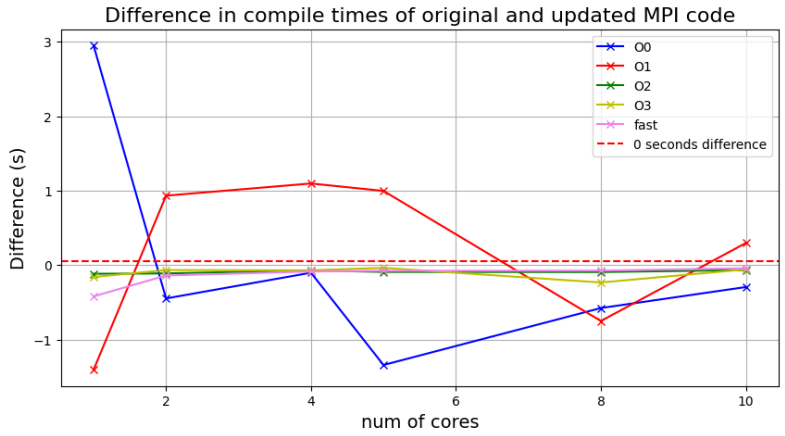
The updated version of the MPI code removes the line that causes the code to abort, the new .c file is called “md-MPI_no_abort.c” in the OneDrive folder. This was done to test whether the line was necessary and could potentially lead to better performance and parallel scalability as this would allow 12 cores to be utilised. However, for all of the numbers of cores that previously caused the code to abort, erroneous results were produced, which can be seen in Appendix A. For the numbers of cores that still produced the result of (-0.09509,-0.16562,49.64602) for the centre of mass, the overall compile time of the code was not effected consistently. There were decreases and increases in compile time for all of these cores, however compile time was not changed consistently, leading to the likelihood that any improvement was just chance, and would be flattened if the code was run multiple times to find an average. The reduction in compile time can be seen to be inconsistent in the graph below.

Code to make .exe:

```
mpiicx -O0 md-MPI_no_abort.c -o md_MPI_no_abort_O0.exe
mpiicx -O1 md-MPI_no_abort.c -o md_MPI_no_abort_O1.exe
mpiicx -O2 md-MPI_no_abort.c -o md_MPI_no_abort_O2.exe
mpiicx -O3 md-MPI_no_abort.c -o md_MPI_no_abort_O3.exe
mpiicx -fast md-MPI_no_abort.c -o md_MPI_no_abort_fast.exe
```

Code to run .exe:

```
for n in `seq 1 12`;do mpirun -np $n ./md_MPI_no_abort_O0.exe | tail -n 2; done
for n in `seq 1 12`;do mpirun -np $n ./md_MPI_no_abort_O1.exe | tail -n 2; done
for n in `seq 1 12`;do mpirun -np $n ./md_MPI_no_abort_O2.exe | tail -n 2; done
for n in `seq 1 12`;do mpirun -np $n ./md_MPI_no_abort_O3.exe | tail -n 2; done
for n in `seq 1 12`;do mpirun -np $n ./md_MPI_no_abort_fast.exe | tail -n 2; done
```



Conclusion

Applying differing levels of intel optimisation to all of the code lead to improvements in compile time, with the best performing optimisations being “-O2” for the serial code and “-O2” and “-O3” performing the best for the other molecular dynamics code. Introducing local variables to the OpenMP lead to consistent improvements in compile time to improve parallel scalability, while updating the MPI code to not abort lead to erroneous results. If the MPI code could be updated to work with any number of cores this would improve parallel scalability. Further optimisation is needed of OpenMP and MPI code to handle the overhead introduced at 7 cores and 8 cores respectively in order to further improve parallel scalability.

References

Intel Corporation (2006) Intel Compilers: Optimizations and Performance Tuning. Available at: https://www.acrc.bris.ac.uk/intel/cc9.1/optaps_cls.pdf (Accessed 15 May 2024)

Big Data

Section 1: Research Task

Research hypothesis

The research hypothesis for this test was “Rides in 2014 starting from Aston Street, Stepney station were shorter compared to other stations”. After testing, this research hypothesis was found to be false.

Data preparation

The data comes in the format of a .zip file, which expands into multiple .csv files. To process this data, a spark session (Spark-3.4.3, 2024) is first initialised. The .csv files are then combined into a single data frame using Spark (code 1, Appendix B). The data frame is then cleared of rows containing any null values, which removes 1239245 rows from the data frame (code 2, Appendix B). In the data frame, each row contains information on a journey from one station to another. Columns of note include “Start date” which contains a string in the format “DD/MM/YYYY HH:MM” which shows when the journey started, “StartStation name” which contains a string that shows which station the journey started from, and “Duration” which contains an integer that shows the duration of the journey in seconds (code 3, Appendix B).

The "Start date” column is turned from a string to a date variable from so that data can be extracted according to months and years. After this, a new data frame is made containing only rows which have a start date containing 2014. This new data frame is the one that will be used for tests as it only contains rows in the time frame relevant to the hypothesis (code 4, Appendix B). Then, a new column is added to this data frame that only contains the month of the starting date of the journey as an integer (code 5, Appendix B).

A loop is then used to examine journeys according to the month they started in, using the new “Month” column that was added. The data is examined by month instead of all together to increase the granularity of the test. Each iteration of the loop creates a new data frame containing journeys that start in that specific month, then information is extracted from the current month’s data frame. This information includes the total number of journeys, the mean journey duration of rides from Aston St, the mean journey duration of rides from all the other stations individually, and the mean journey duration of rides from all the other stations combined. The total number of journeys for a month is found by adding up the total rows for that month’s data frame, and the mean journey durations are found by using the “mean” function from Spark on the “Duration” column of that month’s data frame. The information for each month is saved in a dictionary (code 6, Appendix B).

Test Design

The null hypothesis is that the average journey duration of rides from Aston St are the same or greater than the average journey duration of rides from other stations, and the alternative hypothesis is the same as the given research hypothesis, that the average journey duration of rides from Aston St are shorter than the average journey duration of rides from other stations.

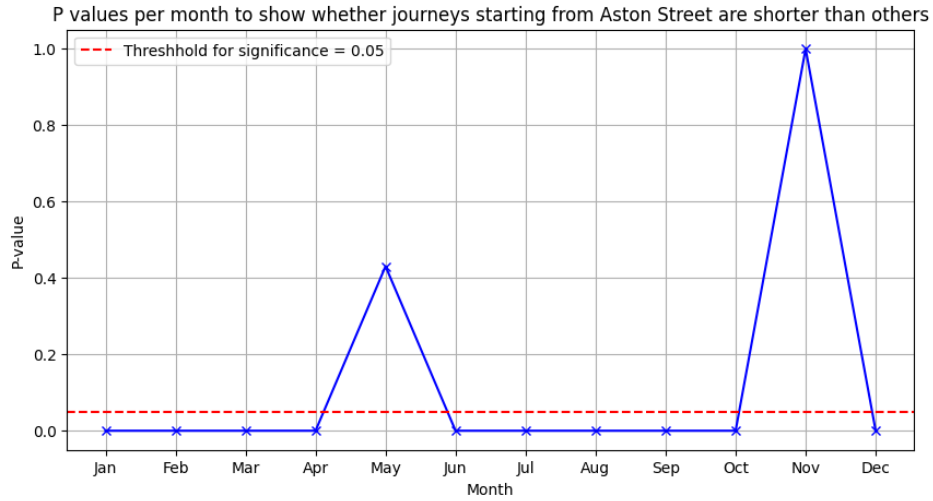
The month specific data is examined individually. For each month, a t-test from the SciPy package (Scipy, 2024) is conducted. Each t-test compares the mean of the sample (the mean journey durations from other stations) to the population mean (the mean journey duration from Aston St).

This returns a p-value, which is the probability that the null hypothesis is true. For each month, a high p-value would indicate that the journey duration from Aston St is not significantly lower than the journey duration from any other station, but a low p-value would indicate that the journey duration from Aston St is significantly lower than the journey duration from other stations. The threshold used for these tests is 0.05, so any p-value of less than 0.05 would lead the null hypothesis to be rejected.

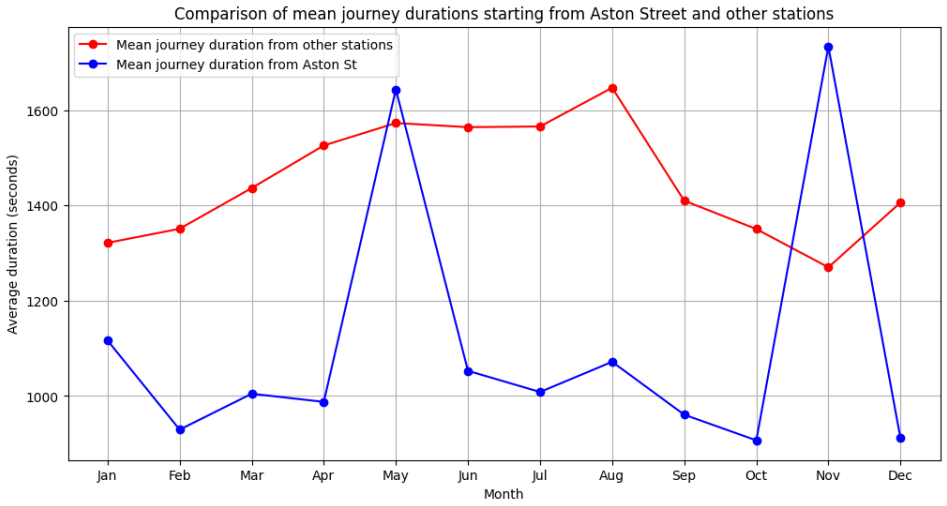
After collecting the p-value for each month (code 7, Appendix B), a weighted mean is taken to determine a single final p-value. This is applied by multiplying the p-value for a given month by the number of total rides in that month, then adding all these values up and dividing the total by the number of total rides taken in the year (code 8, Appendix B).

Results

When examining each month individually, 10 out of 12 months produced a p-value lower than the threshold, which would lead to the null hypothesis being rejected. This can be seen in the graph below, displaying the p-values for each month plotted against the threshold.



In the months where the p-value is higher than the threshold, the average journey duration from Aston St is higher than the average duration from other stations, which can be seen in the graph below.



However, when the weighted mean is taken of all the p-values, the result is 0.08578. The combined p-value is 0.08579, which is close to but not below the threshold of 0.05, which means that the null hypothesis is accepted (code 8, Appendix B). This means that the research hypothesis that "Rides in 2014 starting from Aston Street, Stepney station were shorter compared to other stations", which was taken to be the alternative hypothesis in this study, has been shown to be false. In conclusion, despite journeys from Aston St being shorter than other stations on average for most months, the difference in journey times is not significant enough to warrant the research hypothesis to be true.

Section 2: Critical Reflection

Introduction

Integrating big data into transportation systems enhances their efficiency and responsiveness, encouraging more people to utilise the transportation system, which contributes to lowering carbon emissions. Implementing big data enables real time adjustments to schedules and routes, optimising resource utilisation and user satisfaction. However, adopting big data in this context raises several legal and ethical considerations that must be addressed.

The carbon emissions per person for an average car journey from London to Glasgow produces 120kg of carbon, while the same journey by bus or train journey produces 58 or 71kg respectively of carbon per person, according to Hickman, R. and Banister, D (2007). On top of this, public transport is increasingly utilising electric powered vehicles which produce 30% less carbon emissions (Zemo, 2024). Encouraging more people to use public transportation systems instead of personal vehicles will lead to lower overall carbon emissions.

Social

Big data provides valuable insights that help transportation systems operate more effectively. Analysing large amounts of data allows optimisation of service times and destinations, which can lead to a more consistent, reliable transportation system. A reliable public transport system encourages more consistent use (Göransson, J. and Andersson, H. 2023), reducing dependency on personal vehicles and contributing to lower carbon emissions. Utilising big data also enables transportation authorities to identify and serve underserved areas more effectively. By pinpointing regions lacking adequate services, planners can adjust transportation networks to ensure that transport in these areas is more accessible, further increasing the usage of the whole transport system (Inturri, G. *et al.* 2021). By improving the reliability, efficiency and reach of public transport systems through big data, more individuals are likely to transition from personal vehicles to the transportation system, reducing use of personal vehicles and lowering carbon emissions.

Legal and Ethical

The implementation of big data in public transportation must comply with legal frameworks like the General Data Protection Regulation (GDPR) in the European Union. These regulations mandate strict data security measures and clear guidelines on data usage, ensuring the protection of personal information (EU, 2024). These regulations also mean that it is crucial for transportation

authorities to be transparent about how they use the data they have. Ensuring that passengers understand the purpose and implications of data collection fosters trust in the transportation system as a whole. It is also essential to consider external implications of big data usage in a transportation system. Changes to a transportation that could be implemented based on findings from data analysis could worsen noise pollution and habitat disruption, which can adversely affect local communities and wildlife (Sordello, R. *et al.* 2020).

Conclusion

Big data holds great potential to improve the reliability and efficiency of a transportation system, which in turn would lower carbon emissions by reducing the need for personal vehicles. However, legal and ethical considerations must be made to ensure that data is being handled correctly and that any changes are not disruptive to an area.

References

Spark-3.4.3 (2024) Index of /Spark/Spark-3.4.3. Available at: https://downloads.apache.org/spark/spark-3.4.3/ (Accessed: 08 May 2024).

Scipy (2024) SciPy. Available at: https://scipy.org/ (Accessed: 08 May 2024).

Hickman, R. and Banister, D. (2007) 'Looking over the horizon: Transport and reduced CO2 emissions in the UK by 2030', *Transport Policy*, 14(5), pp. 377–387. doi:10.1016/j.tranpol.2007.04.005.

Zemo (2024) Ultra-Low Emission Bus Scheme, Ultra-Low Emission Bus Scheme | Zemo Partnership. Available at: https://www.zemo.org.uk/work-with-us/buses-coaches/low-emission-buses/ultra-low-emission-bus-scheme.htm (Accessed: 08 May 2024).

Göransson, J. and Andersson, H. (2023) 'Factors that make public transport systems attractive: A review of travel preferences and travel mode choices', *European Transport Research Review*, 15(1). doi:10.1186/s12544-023-00609-x.

Inturri, G. *et al.* (2021) 'Linking public transport user satisfaction with service accessibility for sustainable mobility planning', *ISPRS International Journal of Geo-Information*, 10(4), p. 235. doi:10.3390/ijgi10040235.

Sordello, R. *et al.* (2020) 'Evidence of the impact of noise pollution on Biodiversity: A Systematic Map', *Environmental Evidence*, 9(1). doi:10.1186/s13750-020-00202-y.

EU (2024) General Data Protection Regulation (GDPR). Available at: https://gdpr-info.eu/ (Accessed: 08 May 2024).

Appendix A

OneDrive Link:

https://stummuac-my.sharepoint.com/:f:/g/personal/21359557_stu_mmu_ac_uk/Ej40QneOOQRJr2LQJ-5n6e0Bc1YfU4XDgbRMnECOS6dQcQ?e=Uo2RuX

Serial (md.c) Compile times:

Serial code compile times

	O0 time	O1 time	O2 time	O3 time	fast time
run 1	128.043	54.5981	19.3145	19.406	21.5107
run 2	127.037	55.5127	19.3575	19.3734	21.5495
run 3	126.706	55.8229	19.4193	19.3971	21.6354
avg	**127.262**	**55.31123**	**19.36377**	**19.39217**	**21.5652**

OpenMP (md-OpenMP.c) Compile times:

Default OpenMP compile times

cores	O0 time	O1 time	O2 time	O3 time	fast time
1	133.616	63.9491	53.1201	19.4609	22.7471
2	70.7755	32.3778	26.7747	9.80451	11.4375
3	45.8933	21.9404	19.0918	6.56185	7.57964
4	34.5189	17.4567	14.7232	4.94395	5.78383
5	28.8671	14.8028	11.9027	3.97045	4.62375
6	23.9223	11.6972	10.2309	3.33557	3.81372
7	27.2791	13.7967	12.8142	5.40852	5.91866
8	24.0029	12.5585	11.7737	4.88943	5.41324
9	21.7895	11.3653	11.3414	4.35444	4.91054
10	21.6014	11.2289	11.0579	3.94486	4.42942
11	19.8367	10.2993	10.2987	3.55751	4.04504
12	18.3847	9.6047	9.37116	3.36495	3.77232

OpenMP code changed (md-OpenMP_12.c)

Code changed:

```
totalMass = 0.0;
#pragma omp parallel for reduction(+:totalMass) //this line was added
for (i=0; i<num; i++) {
        totalMass += mass[i];
}
```

```
/*rest of code*/

#pragma omp parallel for default(none) \
  private(i,j,dx,dy,dz,temp_d,d,F,ax,ay,az)      \
  shared(vx,vy,vz,num,old_x,old_y,old_z,x,y,z,mass,old_mass,GRAVCONST)
    for(i=0; i<num; i++) {
      double vxLocal = 0.0; //added local variables
      double vyLocal = 0.0; //added
      double vzLocal = 0.0; //added
      // calc forces on body i due to particles (j != i)
      for (j=0; j<num; j++) {
        if (j != i) {
          dx = old_x[j] - x[i];
          dy = old_y[j] - y[i];
             dz = old_z[j] - z[i];
          temp_d =sqrt(dx*dx + dy*dy + dz*dz);
          d = temp_d>0.01 ? temp_d : 0.01;
          F = GRAVCONST * mass[i] * old_mass[j] / (d*d);
             // calculate acceleration due to the force, F
          ax = (F/mass[i]) * dx/d;
          ay = (F/mass[i]) * dy/d;
             az = (F/mass[i]) * dz/d;
             // approximate LOCAL velocities in "unit time"
          vxLocal += ax; //updates the values locally
          vyLocal += ay; //added
          vzLocal += az; //added
        }
      }
      //update GLOBAL velocities
      vx[i] += vxLocal; //updates values globally
      vy[i] += vyLocal; //added
      vz[i] += vzLocal; //added

      // calc new position
      x[i] = old_x[i] + vx[i];
      y[i] = old_y[i] + vy[i];
      z[i] = old_z[i] + vz[i];

    } // end of LOOP 2
```

Updated OpenMP (md-OpenMP_12.c) Compile times:

updated OpenMP compile times

cores	O0 time	O1 time	O2 time	O3 time	fast time
1	115.683	30.3666	19.6129	19.6756	21.763
2	58.0528	15.3187	9.88633	9.87614	10.9526
3	38.6999	10.2455	6.61076	6.63956	7.28962
4	29.0918	7.69634	5.00383	4.98194	5.49711
5	23.3631	6.19289	3.95994	3.99967	4.40376
6	19.4978	5.27018	3.3429	3.33765	3.6845
7	24.0972	7.56351	5.43473	5.45527	5.86068
8	21.2903	6.86231	4.91285	4.87997	5.30295
9	19.7318	6.33235	4.3744	4.38261	4.83884
10	19.4721	5.87787	3.9705	3.97715	4.39363
11	17.7594	5.4023	3.60458	3.58858	3.97936
12	16.3172	4.95891	3.32041	3.37147	3.68306

Difference in OpenMP and Updated OpenMP compile times

difference in OpenMP compile times (original minus updated)					
cores	O0 dif	O1 dif	O2 dif	O3 dif	fast dif
1	17.933	33.5825	33.5072	-0.2147	0.9841
2	12.7227	17.0591	16.88837	-0.07163	0.4849
3	7.1934	11.6949	12.48104	-0.07771	0.29002
4	5.4271	9.76036	9.71937	-0.03799	0.28672
5	5.504	8.60991	7.94276	-0.02922	0.21999
6	4.4245	6.42702	6.888	-0.00208	0.12922
7	3.1819	6.23319	7.37947	-0.04675	0.05798
8	2.7126	5.69619	6.86085	0.00946	0.11029
9	2.0577	5.03295	6.967	-0.02817	0.0717
10	2.1293	5.35103	7.0874	-0.03229	0.03579
11	2.0773	4.897	6.69412	-0.03107	0.06568
12	2.0675	4.64579	6.05075	-0.00652	0.08926

MPI (md-MPI.c) Compile times:

Default MPI compile times					
cores	O0 time	O1 time	O2 time	O3 time	fast time
1	130.183	55.8353	19.4124	19.2831	22.7006
2	63.4072	30.917	9.68788	9.71462	11.4197
3	ABORT	ABORT	ABORT	ABORT	ABORT
4	33.3528	14.962	4.90651	4.88056	5.75482
5	25.632	12.1573	3.9132	3.94776	4.58709
6	ABORT	ABORT	ABORT	ABORT	ABORT
7	ABORT	ABORT	ABORT	ABORT	ABORT
8	25.8437	13.6977	4.90646	4.94746	5.5806
9	ABORT	ABORT	ABORT	ABORT	ABORT
10	20.5067	11.6035	3.9552	3.95834	4.44446
11	ABORT	ABORT	ABORT	ABORT	ABORT
12	ABORT	ABORT	ABORT	ABORT	ABORT

MPI code changed (md-MPI_no_abort.c)

Code changed:

// this code only handles num/numPEs being integer value
// if ((num/numPEs)*numPEs != num) MPI_Abort(MPI_COMM_WORLD, -1); //this was the line that was changed - it was only changed to be removed

Updated MPI compile times and erroneous results (left) and difference in compile times of original MPI and Updated MPI code (right):

Updated MPI compile times

cores	O0 time	O1 time	O2 time	O3 time	fast time	Erroneous answers:
1	127.232	57.2408	19.5283	19.4422	23.1202	N/A
2	63.8545	29.9835	9.79881	9.78063	11.5588	N/A
3	43.95	19.5188	6.5587	6.60478	7.71402	(-0.09523,-0.16592,49.64396)
4	33.4545	13.865	4.97762	4.95047	5.83995	N/A
5	26.9733	11.1601	4.00422	3.98395	4.66319	N/A
6	21.5378	10.5366	3.33481	3.34888	3.8604	(-0.09523,-0.16592,49.64396)
7	30.1278	16.0872	5.70915	5.68471	6.50728	(-0.09522,-0.16626,49.64366)
8	26.4199	14.4479	4.99891	5.18038	5.65395	N/A
9	23.1322	12.3584	4.43494	4.48453	4.98412	(-0.09523,-0.16592,49.64396)
10	20.8007	11.3062	4.02116	4.01507	4.48602	N/A
11	19.068	10.1929	3.66748	3.67963	4.07739	(-0.09523,-0.16592,49.64396)
12	17.4965	9.62614	3.38333	3.38035	3.82362	(-0.09616,-0.16281,49.63728)

difference in MPI compile times (original minus updated)

cores	O0 dif	O1 dif	O2 dif	O3 dif	fast dif
1	2.951	-1.4055	-0.1159	-0.1591	-0.4196
2	-0.4473	0.9335	-0.11093	-0.06601	-0.1391
3					
4	-0.1017	1.097	-0.07111	-0.06991	-0.08513
5	-1.3413	0.9972	-0.09102	-0.03619	-0.0761
6					
7					
8	-0.5762	-0.7502	-0.09245	-0.23292	-0.07335
9					
10	-0.294	0.2973	-0.06596	-0.05673	-0.04156
11					
12					

Appendix B

OneDrive Link:

https://stummuac-my.sharepoint.com/:f:/g/personal/21359557_stu_mmu_ac_uk/Ej40QneOOQRJr2LQJ-5n6e0Bc1YfU4XDgbRMnECOS6dQcQ?e=Uo2RuX

Code:

1)

```
df = spark.read.csv("/content/Datasets/*.csv", header=True) #adding all csvs to a dataframe
df.show()
```

```
+---------+--------+-------+----------------+------------+--------------------+----------------+--------------+--------------------+
|Rental Id|Duration|Bike Id|        End Date|EndStation Id|     EndStation Name|      Start Date|StartStation Id|   StartStation Name|
+---------+--------+-------+----------------+------------+--------------------+----------------+--------------+--------------------+
| 34263367|    1080|   9076|24/06/2014 00:57|         695|Islington Green, ...|24/06/2014 00:39|           311|Foley Street, Fit...|
| 34603487|     660|   6328|03/07/2014 11:51|         695|Islington Green, ...|03/07/2014 11:40|            22|Northington Stree...|
| 34689078|     120|   2006|05/07/2014 15:09|         357|Howland Street, F...|05/07/2014 15:07|           311|Foley Street, Fit...|
+---------+--------+-------+----------------+------------+--------------------+----------------+--------------+--------------------+
```

2)

```
no_nulls_df = df.dropna(how='any', thresh=None, subset=None) #creating dataframe with any rows with any null values removed
num_of_nulls = df.count() - no_nulls_df.count() #calculating how many rows contained null values
print("number of nulls: ", num_of_nulls) #printing total null containing rows
```

```
number of nulls:  1239245
```

3)

```
no_nulls_df = no_nulls_df.withColumn("Start Date", to_timestamp(col("Start Date"), "dd/MM/yyyy HH:mm"))
no_nulls_df.show()
```

```
+---------+--------+-------+----------------+------------+--------------------+-------------------+--------------
|Rental Id|Duration|Bike Id|        End Date|EndStation Id|     EndStation Name|         Start Date|StartStation Id
+---------+--------+-------+----------------+------------+--------------------+-------------------+--------------
| 34263367|    1080|   9076|24/06/2014 00:57|         695|Islington Green, ...|2014-06-24 00:39:00|           311
| 34603487|     660|   6328|03/07/2014 11:51|         695|Islington Green, ...|2014-07-03 11:40:00|            22
```

4)

```
in_2014_df = no_nulls_df.filter(year(col("Start Date")).contains(2014))
in_2014_df.show()
```

```
+---------+--------+-------+----------------+------------+--------------------+-------------------+--
|Rental Id|Duration|Bike Id|        End Date|EndStation Id|     EndStation Name|         Start Date|S
+---------+--------+-------+----------------+------------+--------------------+-------------------+--
| 34263367|    1080|   9076|24/06/2014 00:57|         695|Islington Green, ...|2014-06-24 00:39:00|
| 34603487|     660|   6328|03/07/2014 11:51|         695|Islington Green, ...|2014-07-03 11:40:00|
| 34689078|     120|   2006|05/07/2014 15:09|         357|Howland Street, F...|2014-07-05 15:07:00|
```

5)

```
in_2014_df = in_2014_df.withColumn("Month", month("Start Date"))
in_2014_df.show()
```

```
+---------+--------+-------+----------------+------------+--------------------+-------------------+--------------+--------------------+-----+
|Rental Id|Duration|Bike Id|        End Date|EndStation Id|     EndStation Name|         Start Date|StartStation Id|   StartStation Name|Month|
+---------+--------+-------+----------------+------------+--------------------+-------------------+--------------+--------------------+-----+
| 34263367|    1080|   9076|24/06/2014 00:57|         695|Islington Green, ...|2014-06-24 00:39:00|           311|Foley Street, Fit...|    6|
| 34603487|     660|   6328|03/07/2014 11:51|         695|Islington Green, ...|2014-07-03 11:40:00|            22|Northington Stree...|    7|
| 34689078|     120|   2006|05/07/2014 15:09|         357|Howland Street, F...|2014-07-05 15:07:00|           311|Foley Street, Fit...|    7|
```

6)

```
months = {} #initialise dictionary to store data for each month

#loops through each month, filters dataframe to that month, filters further into Aston ST and not Aston St, calculate means, store results
for i in range(1,13): #1-13 so that it starts from 1 and ends on 12, how the months are written in df

    months[i] = {} #initialise dictionary to store results for current month

    month_df = in_2014_df.filter(col("Month").contains(i)) #filter df for current month

    months[i]["total_rides"] = month_df.count() #show total rides for month (used for weighted avg)

    aston_df = month_df.filter(col("StartStation Name") == "Aston Street, Stepney")
    others_df = month_df.filter(col("StartStation Name") != "Aston Street, Stepney") #filter df for Aston Street and other stations

    aston_mean_df = aston_df.agg(mean("Duration").alias("average_duration"))#Calc mean duration for aston
    months[i]["aston_avg_duration"] = aston_mean_df.collect()[0]["average_duration"] #add to dictionary for current month

    others_means_df = others_df.groupBy("StartStation Id").agg(mean("Duration").alias("average_duration")) #groups other stations + finds mean for each
    others_means_collected = others_means_df.collect() #add means for other stations to a list
    months[i]["others_means"] = [row['average_duration'] for row in others_means_collected] # adds list of means for other stations to dictionary for current month

    others_avg_duration_df = others_df.agg(mean("Duration").alias("average_duration")) #calc mean duration from other stations (only for graph)
    months[i]["others_avg_duration"] = others_avg_duration_df.collect()[0]["average_duration"] #add to dic.

    print(i, " done")
```

7)

```
for month in months:
  # - t test -
  #null: journey durations from other stations are greater than journey durations from Aston street
  #alternate: journeys durations from other stations are greater than from Aston (ie journeys from Aston are shorter)
  #p val threshhold: 0.05 -> if higher reject null hypothesis
    months[month]["t_stat"], months[month]["p_val"] = stats.ttest_lsamp(months[month]["others_means"], months[month]["aston_avg_duration"], alternative="greater")

    print("Month: ", month) #outputting results
    print("Total rides: ", months[month]["total_rides"])
    print("Aston mean jounrey duration:", months[month]["aston_avg_duration"])
    print("Mean journey duration from other stations ", months[month]["others_avg_duration"])
    print("T stat:", months[month]["t_stat"])
    print("P val: ", months[month]["p_val"])
    print(" ")
```

8)

```
#calc weighted avg p value
total_rides_year = 0
p_vals_weighted = 0

for month in months:
  total_rides_year += months[month]["total_rides"] #holds total rides in the year
  p_vals_weighted += (months[month]["p_val"] * months[month]["total_rides"]) #adds up all the p values multiplied the rides for that
p_vals_weighted_avg = p_vals_weighted / total_rides_year #divide total p val by total rides to get weighted avg

if p_vals_weighted_avg < 0.05: #messages for whether or not Null is accepted
  print("Alternate hypothesis accepted, p value: ", p_vals_weighted_avg, " is less than 0.05 threshold")
else:
  print("Null hypothesis accepted, weighted average p value: ", p_vals_weighted_avg, " is greater than 0.05 threshold")
```

```
Null hypothesis accepted, weighted average p value:  0.08578904717860927  is greater than 0.05 threshold
```