HPC-BD ASSESSMENT

NAME: RAMSES MORENO DE LA CRUZ

MMU ID: 23624012

DECLARATION:

A signed plagiarism and ethics disclaimer, as follows: 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. It has been undertaken in accordance with the University research ethics standards.

Signed: Ramses Moreno De La Cruz

<u>BIG DATA ASSESSMENT</u>

1. Introduction

This project aims to examine, analyze, and gain insights into the cycle hire usage in London during 2014 and consequently determine if the provided research hypothesis is true. In addition, we will discuss the role of big data in achieving a more sustainable global environment.

2. Data Analysis

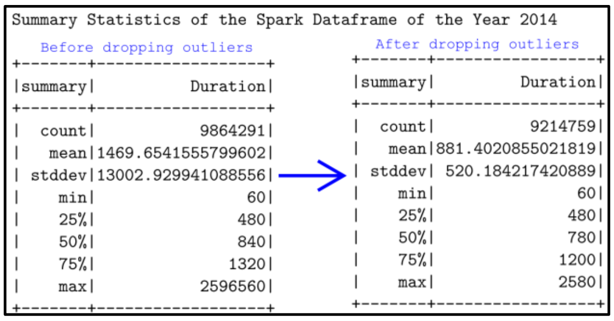
The process to analyze the provided dataset of cycle hire usage in London during 2014 started with setting up the libraries and respective modules. Then the dataset was downloaded and unzipped. After that, the following stages were executed:

<u>Stage 1</u>: The data files were checked and then joined into a single main Spark dataframe with 11481596 rows.

<u>Stage 2</u>: The duplicated rows (1239112 rows, i.e., 10.79% of the data) and the rows with "Duration = 0" (32503 rows, i.e., 0.28% of the data) were dropped.

<u>Stage 3</u>: Then the rows that do not belong to the year 2014 were dropped (345690 rows, i.e., 3.01% of the data).

<u>Stage 4</u>: Then the rows of the column "Duration" that were considered outliers according to the Interquartile Range were dropped (2266837 rows, i.e., 19.74% of the data). As a consequence, the process became computationally less expensive and significant changes happened in the mean, standard deviation, and quantiles (see Image 1), which impacted the research hypothesis test.



```
Summary Statistics of the Spark Dataframe of the Year 2014
   Before dropping outliers                   After dropping outliers
+-------+------------------+             +-------+-----------------+
|summary|          Duration|             |summary|         Duration|
+-------+------------------+             +-------+-----------------+
|  count|           9864291|             |  count|          9214759|
|   mean|1469.6541555799602|             |   mean|881.4020855021819|
| stddev|13002.929941088556|  ------->   | stddev| 520.184217420889|
|    min|                60|             |    min|               60|
|    25%|               480|             |    25%|              480|
|    50%|               840|             |    50%|              780|
|    75%|              1320|             |    75%|             1200|
|    max|           2596560|             |    max|             2580|
+-------+------------------+             +-------+-----------------+
```

Image 1. Summary statistics of the dataset for the year 2014

<u>Stage 5</u>: Graphs and statistical summaries of the rides for the year 2014, per station and per season, were made (see Image 2).

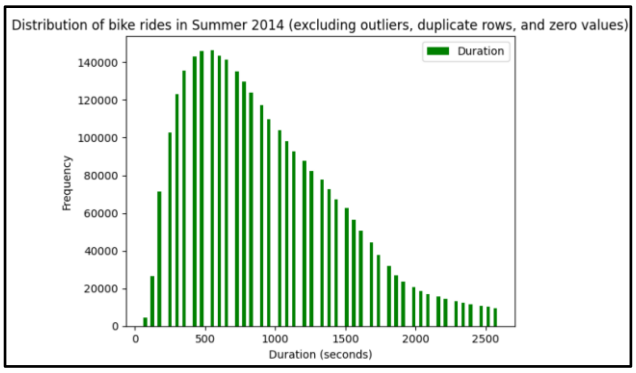


Image 2. Distribution of bike rides in Summer 2014

<u>Stage 6</u>: The final dataframe (9214759 rows) was split into one subset for the "Station Baylis Road, Waterloo station" and another subset for the rest of the stations. Then, each subset was filtered and grouped with the purpose of obtaining the mean duration value per month per station. Finally, they were transformed from Spark to Pandas dataframe and then into Series to apply the research hypothesis tests using Python (SciPy). In other words, this stage ends with 2 groups of data that contain only the "mean duration value per month per station" (where group 1 consists of Baylis Road and Waterloo station, and group 2 includes the other stations).

<u>Stage 7</u>: The following tests were executed to research the hypothesis that "*Rides in 2014 starting from Baylis Road, Waterloo station were shorter compared to other stations*":

- <u>Levene's test</u>: This is the first step of the testing strategy. This test checks an assumption of another statistical test. In this case, the other statistical test is the "T-test for the means of two independent samples". Levene's test is useful for comparing whether two or more groups of samples (where group 1 represents the mean duration value per month of Baylis Road, Waterloo station; and group 2 represents the mean duration value per month of the other stations) have equal variance. The hypotheses are as follows:
    - Null hypothesis (H0): The groups of samples being compared have the same variances or their variances are not significantly different.
    - Alternative hypothesis (H1): The groups of samples being compared have significantly different variances.
- <u>T-test for the means of two independent samples</u>: This is the second step of the testing strategy. This test is convenient for comparing the means of two groups of samples (where group 1 represents the mean duration value per month of Baylis Road, Waterloo station; and group 2 represents the mean duration value per

month of the other stations). The hypotheses for this test are as follows:

- Null hypothesis (H0): The two samples being compared have the same mean or their means are not significantly different.
- Alternative hypothesis (H1): The mean of the first sample (i.e., Baylis Road, Waterloo station) is less than the mean of the distribution underlying the second sample (i.e., the other stations).

3. Insights

The output of research hypothesis tests was as follows:

- p-value, Levene's test: 0.002597429565560061
- p-value, T-test for the means of two independent samples: 1.5987644109689258e-07

In the context of considering a significance level of 5% or 0.05 for comparing the probability value (i.e., the p-value) and determining if there is strong evidence to reject the H0 or not:

- The Levene's test provided evidence to reject the Null Hypothesis (p-value < 0.05). It means that the variance of the data of Baylis Road, Waterloo station is significantly different from the variance of the data of the other stations.
- ***The T-test for the means of two independent samples provided strong evidence to reject the Null Hypothesis (p-value < 0.05) and statistically conclude that the rides in 2014 starting from Baylis Road, Waterloo station were shorter compared to other stations***. Finally, it is important to remark that the data provided was enough to obtain a conclusive result.

Other findings after digging into the dataset (before dropping the outliers) are as follows:

- The Baylis Road, Waterloo station is the station in the third position from which most rides started and finished in the year 2014 (78295 and 80818 rides respectively), meanwhile the first place had 99622 and 99265 rides respectively.
- The mean duration of a ride per season in 2014 were: Winter (1321.81 seconds), Autumn (1356.78 seconds), Spring (1519.98 seconds), and Summer (1596.03 seconds); which is consistent with the behavior patterns of the people due to weather conditions.
- The mean duration of a ride in 2014 was 1469.65 seconds, meanwhile for the Baylis Road, Waterloo station was 1046.68 seconds.

4. Recommendations

Some recommendations for Transport for London (TfL) to improve the service and increase the number of users are as follows:

- Enhance the signs for cyclists (road markings, traffic lights, messages at road junctions, and minimize gaps in cycle lanes).
- Adjust the fares or even apply the scheme of dockless bikes to compete with private dockless bikes.
- Develop more cycle lanes in the boroughs nearby to the center of London.
- If possible, establish more car-free zones or expand the existing car-free zones as suggested by the Greater London Authority 2024 (Mayor of London, 2021).

5. Critical Reflection

Nowadays, the role of big data in achieving a more sustainable global environment is extremely complex, as it will be described in the following paragraphs.

The use of big data has had well-known and positive impacts in various sectors over the past 15 to 20 years, including banking, healthcare, manufacturing, transportation, logistics, marketing (Ye, 2024), academia, sports, and many other diverse industries. However, the numerous benefits in the short or medium term may be overshadowed by the significant amount of pollution, specifically the greenhouse gas emissions, associated with some or all components of the complex and puzzling ecosystem that encompasses the concept of big data. From the design of software with increasingly complex algorithms to the manufacturing, shipping, and use of hardware such as data centers, computers, and cellphones (which are becoming more affordable every day), everything should be taken into account when assessing the significant role of big data in achieving a more sustainable global environment.

From a legal standpoint, it is undeniable that big data is driving world, business, local, and community leaders to play an active role in creating and implementing new policies to change the alarming direction our society is heading in terms of the environment. However, strict or extreme environmental policies (influenced by big data) could be a dangerous path for our society if they lead us to a world where almost every aspect of human life is monitored and measured in order to reduce carbon emissions. In other words, there are numerous ethical concerns, such as privacy, and boundaries that need to be established before pushing the world towards the goal of zero emissions.

Probably, like many other challenges faced by human beings throughout the history of mankind, the development of better tools or technologies and an enhanced understanding of our environment will lead to a future in which not only the goal of zero emissions is achieved, taking into account the respective legal, ethical, and social concerns, but also reverses the detrimental effects that we have caused to the entire planet so far.

For example, the use of big data to efficiently manage the energy network of building (Guzhov and Krolin, 2018), to predict the weather with high accuracy (Alam and Amjad, 2019), and even to change the behavioral patterns of people (as is the case with many citizens of London who have been using bikes instead of cars for the past 10 years, resulting in improved health, reduced pollution in the city, and many other positive side effects due to the use of big data) could be replicated and have the potential to progressively and sustainably change the planet.

HIGH PERFORMANCE COMPUTING ASSESSMENT

1. Implementation of the code

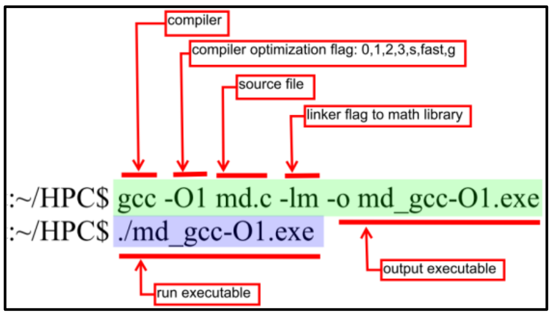
Firstly, the Serial and OpenMP code of this project was compiled exclusively using the GNU Compiler Collection (GCC), while the MPI code was compiled using mpicc.

Furthermore, the Serial, OpenMP, and MPI code was compiled and run on the department's Linux machines, which have the following features:

- CPU(s): 12
- Thread(s) per core: 2
- Core(s) per socket: 6
- Socket(s): 1
- Model: 158
- Model name: Intel(R) Core(TM) i7-8700 CPU @ 3.20GHz

1.1. Serial Code

The serial source code for the simulation of molecular dynamics was compiled and run applying the following command structure, shadowed in green and blue respectively (see Image 1).



```
:~/HPC$ gcc -O1 md.c -lm -o md_gcc-O1.exe
:~/HPC$ ./md_gcc-O1.exe
```

compiler
compiler optimization flag: 0,1,2,3,s,fast,g
source file
linker flag to math library
output executable
run executable

Image 1. Serial Code (compile and run)

It is important to highlight that the main differences between the several compilations and runs undertaken were the compiler optimization flags (O0, O1, O2, O3, Os, Ofast, and Og) (Free Software Foundation, Inc., no date) used in every compilation and consequently the name of the executable generated after every compilation.

The methodology for determining the time taken was supported by the Linux header "include <sys/time.h>" and its respective function "gettimeofday()" as a way to get and compute the time before and after completing all time-steps.

The output of the runs undertaken (for every optimization flag) is shown in the Table 1:

Serial Code Results		
Condition: GCC Compiler		
Optimization Flags	**Timing (seconds)**	**Speed-Up (seconds / seconds)**
O0	135.104	1.0
O1	78.4364	1.7
O2	61.0536	2.2
O3	67.6191	2.0
Os	69.5574	1.9
Ofast	53.1427	2.5
Og	81.7103	1.7

Table 1. Serial code results

It is essential to notice that the center of mass (shown in Table 2) at the end of the running process of every single compiler optimization flag (O0, O1, O2, O3, Os, Ofast, and Og) was the same.

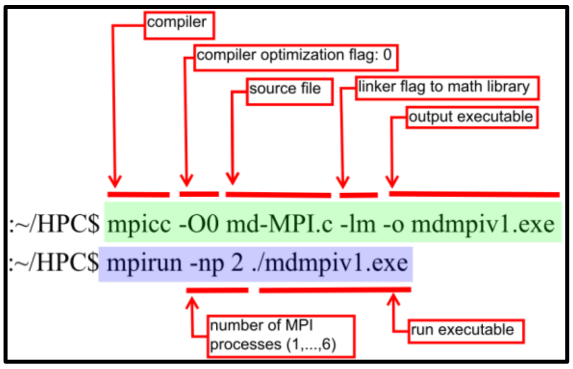
Centre of mass	(-0.09509,-0.16562,49.64602)

Table 2. Centre of mass (Serial code)

1.2. MPI Code

The source code for the simulation of molecular dynamics was compiled and run with the optimization flags O0 (baseline), O2, and Ofast. The latter and former were part of the top 3 optimization flags (in terms of performance) according to the results of the Serial code.

The MPI source code for the MPI processes from 1 to 6 was compiled and run using the following commands' structure shadowed in green and blue respectively (see Image 2).

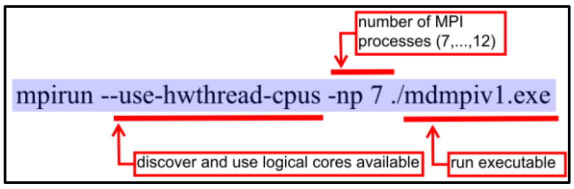


```
:~/HPC$ mpicc -O0 md-MPI.c -lm -o mdmpiv1.exe
:~/HPC$ mpirun -np 2 ./mdmpiv1.exe
```

compiler
compiler optimization flag: 0
source file
linker flag to math library
output executable
number of MPI processes (1,...,6)
run executable

Image 2. MPI Code for 1 to 6 MPI processes (compile and run)

The MPI source code for the MPI processes from 7 to 12 was run using the following command structure, which is highlighted in blue (see Image 3). The purpose was to determine the number of logical cores offered by the machine due to hyperthreading technology (The Open MPI Project, no date), and then utilize them. In this case, the Linux machine in the department has 12 logical cores.



```
mpirun --use-hwthread-cpus -np 7 ./mdmpiv1.exe
```

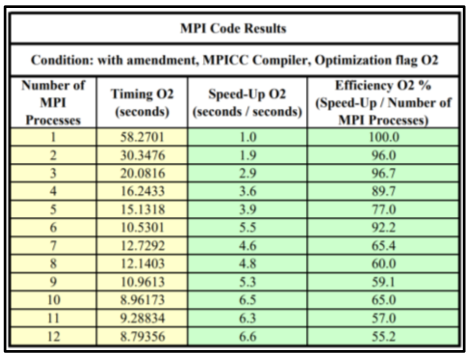
number of MPI processes (7,...,12)
discover and use logical cores available
run executable

Image 3. MPI Code for 7 to 12 MPI processes (run)

It is important to notice that the compilation process was performed only once, and the main differences between the runs undertaken were the number of MPI processes executed (from 1 to 12).

The methodology for determining the time taken was supported by the Linux header "include <sys/time.h>" and its respective function "gettimeofday()" as a way to get and compute the time before and after completing all time-steps.

The best output of the runs undertaken (i.e., flag O2) is shown in Table 3.



MPI Code Results			
Condition: with amendment, MPICC Compiler, Optimization flag O2			
Number of MPI Processes	Timing O2 (seconds)	Speed-Up O2 (seconds / seconds)	Efficiency O2 % (Speed-Up / Number of MPI Processes)
1	58.2701	1.0	100.0
2	30.3476	1.9	96.0
3	20.0816	2.9	96.7
4	16.2433	3.6	89.7
5	15.1318	3.9	77.0
6	10.5301	5.5	92.2
7	12.7292	4.6	65.4
8	12.1403	4.8	60.0
9	10.9613	5.3	59.1
10	8.96173	6.5	65.0
11	9.28834	6.3	57.0
12	8.79356	6.6	55.2

Table 3. MPI code results (for optimization flag O2).

It is essential to notice that the center of mass, at the end of the running process of the MPI process (no matter if the optimization flag was O0, O2, or Ofast), differs in the following way (see Table 4):

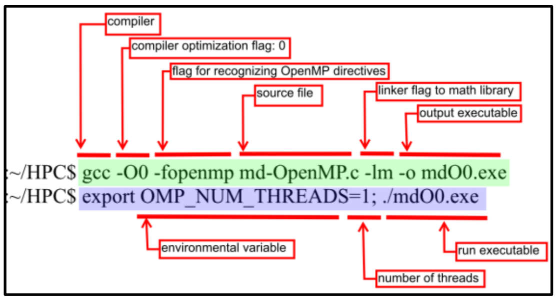
MPI Process	Centre of mass
1, 2, 4, 5, 8, 10	(-0.09509,-0.16562,49.64602)
3, 6, 9, 11	(-0.09523,-0.16592,49.64396)
7	(-0.09522,-0.16626,49.64366)
12	(-0.09616,-0.16281,49.63728)

Table 4. Centre of mass (MPI code) for optimization flag O0, O2, and Ofast

1.3. OpenMP Code

The source code for the simulation of molecular dynamics was compiled and run with the optimization flags O0 (baseline), O3, and Ofast. The latter and former were part of the top 3 optimization flags (in terms of performance) according to the results of the Serial code.

The OpenMP source code for all number of threads was compiled and run using the following command structure shadowed in green and blue respectively (see Image 4).



```
~/HPC$ gcc -O0 -fopenmp md-OpenMP.c -lm -o mdO0.exe
~/HPC$ export OMP_NUM_THREADS=1; ./mdO0.exe
```

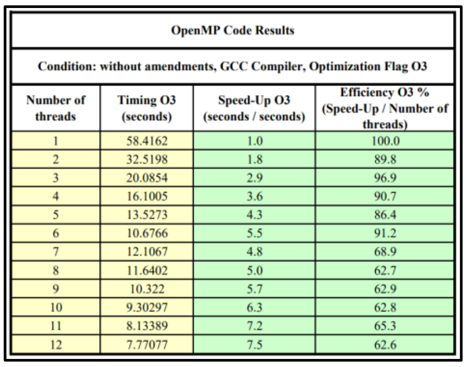
compiler
compiler optimization flag: 0
flag for recognizing OpenMP directives
source file
linker flag to math library
output executable
environmental variable
run executable
number of threads

Image 4. OpenMP Code (compile and run)

It is important to notice that the compilation process was performed only once, and the main differences between the runs undertaken were the number of threads used (from 1 to 12).

The methodology for determining the time taken was supported by the OpenMP function "omp_get_wtime()" as a way to get the time elapsed during the process of completing all time-steps (OpenMP Architecture Review Board, no date).

The best output of the runs undertaken (i.e., flag O3 is shown in Table 5.



OpenMP Code Results			
Condition: without amendments, GCC Compiler, Optimization Flag O3			
Number of threads	Timing O3 (seconds)	Speed-Up O3 (seconds / seconds)	Efficiency O3 % (Speed-Up / Number of threads)
1	58.4162	1.0	100.0
2	32.5198	1.8	89.8
3	20.0854	2.9	96.9
4	16.1005	3.6	90.7
5	13.5273	4.3	86.4
6	10.6766	5.5	91.2
7	12.1067	4.8	68.9
8	11.6402	5.0	62.7
9	10.322	5.7	62.9
10	9.30297	6.3	62.8
11	8.13389	7.2	65.3
12	7.77077	7.5	62.6

Table 5. OpenMP code results (for optimization flag O3).

It is essential to notice that the center of mass (shown in Table 6) at the end of the running process of every thread was the same (no matter if the optimization flag was O0, O3, or Ofast).

Centre of mass	(-0.09509,-0.16562,49.64602)

Table 6. Centre of mass (OpenMP code) for optimization flag O0, O3, and Ofast

2. Performance of the code

2.1. Serial Code

The insights into the performance of the Serial code are as follows:

- Functionality: There is a clear tradeoff between the time to complete all time-steps (see column "Timing" in Table 1) and the modifications made to the source as a consequence of the optimization flags during the compilation process. It is relevant to notice that those changes in the source code are made with the aim to enhance the performance of it (decrease "timing"), but the downside is that they sometimes make debugging activities daunting (Free Software Foundation, Inc., no date).
- Numeric: In this spectrum of tradeoffs (see Table 7), the results show that the timing decreases from no optimization (flag O0 with 135.104 seconds) to maximum optimization (flag Ofast with 53.1427 seconds). In other words, the timing with the optimization flag

"Ofast" is 2.5 times faster than with the optimization flag "O0".

Optimization Flags' Spectrum
O0 → Og → O1 → Os → O3 → O2 → Ofast
More execution time → Less execution time
Easy to debug → Difficult to debug

Table 7. Optimization Flags Spectrum

2.2. MPI Code

The insights into the performance of the MPI code are as follows:

- Functionality: the implementation of MPI in this hardware context (i.e., in only one single node) does not appear to be useful, considering that it works well when applied in a network of nodes (i.e., distributed-memory systems). As a result, the "timing" increased and the "efficiency" decreased compared to OpenMP.
- Numeric: The code run using the optimization flags O0 and Ofast showed evidence of efficiency rebounds (see Table 3) or even efficiency levels above 100% (see Image 5) in some of the first 6 MPI processes (i.e., an event called super-linear speedup). The super-linear speedup used to happen mainly due to the increase in the cache resources in parallel computer architectures (Ristov et al., 2016). In other words, as the problem of molecular dynamics remained fixed, the number of processors increased. Consequently, the aggregate cache allowed the appearance of the super-linear speedup, but only in a temporary way (see Image 5), because other factors such as overhead (related to communication and synchronization) and contentions (associated with the current shared-memory situation) decrease the efficiency.
- In addition, it is important to notice the sharp decrease in efficiency in the transition from MPI process number 6 to MPI process number 7 (a drop in efficiency around 20% to 40% depending on the optimization flag used), which is reasonable considering that the MPI processes started sharing the physical cores at that moment (i.e., contention or competition for access to the shared memory).

2.3. OpenMP Code

The insights into the performance of the OpenMP code are as follows:

- Functionality: Due to the fact that properties related to scalability (i.e., using multiple nodes) are not required in this simulation of molecular dynamics, the OpenMP code, which excels at parallelism in shared-memory systems (i.e., within a single node), proves to be a reliable option. However, like any other kind of parallel implementation, it is affected by some overheads (related to synchronization, thread creation, and thread management) and contentions (associated with the current shared-memory situation).
- Numeric: There was a great performance of the OpenMP code (timing, speed-up, and efficiency) compared to the Serial code and MPI code. Indeed, the efficiency of the OpenMP remained above 60% across the 12 threads. This outstanding performance is expected, considering that the OpenMP code works with more than one core (something that the Serial code was not able to do) and it works well in shared-memory systems (something that the MPI code struggled to do).
- In addition, it is important to notice the decrease in efficiency during the transition from thread 6 to thread 7 (a drop in efficiency around 20% to 30% depending on the optimization flag used). This decrease is moderate, considering that the threads began to share the physical cores at that moment, leading to contention or competition for access to the shared memory.

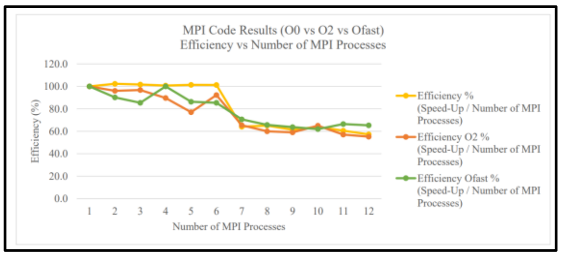
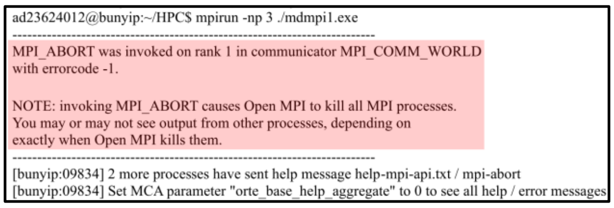


Image 5. OpenMP, optimization flag O0 (Efficiency vs Number of threads)

3. Amended Code

3.1. MPI Code

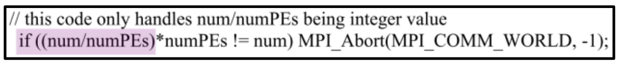
It is crucial to notice that when running the compiled source file with 3 and 6 MPI processes (using the compiler optimization flag O0), an error message emerged (see Image 6). The original MPI source code was edited in the specific section (highlighted in purple, see Image 7) with the aim to allow the user to implement the source code in the totality of MPI processes.



```
ad23624012@bunyip:~/HPC$ mpirun -np 3 ./mdmpi1.exe
--------------------------------------------------------------------------
MPI_ABORT was invoked on rank 1 in communicator MPI_COMM_WORLD
with errorcode -1.

NOTE: invoking MPI_ABORT causes Open MPI to kill all MPI processes.
You may or may not see output from other processes, depending on
exactly when Open MPI kills them.
--------------------------------------------------------------------------
[bunyip:09834] 2 more processes have sent help message help-mpi-api.txt / mpi-abort
[bunyip:09834] Set MCA parameter "orte_base_help_aggregate" to 0 to see all help / error messages
```

Image 6. Error message



```
// this code only handles num/numPEs being integer value
 if ((num/numPEs)*numPEs != num) MPI_Abort(MPI_COMM_WORLD, -1);
```

Image 7. Edited section of the original MPI code

The error message MPI_ABORT was invoked because the line in the MPI source code that checks if the total number of molecules (represented by the variable "num") is evenly divisible by the number of MPI processes (represented by the variable "numPEs") spotted that this condition was not met (i.e., the number of molecules could not be evenly distributed among the MPI processes). Finally, it is important to remark that this code amendment did not change the output of the run undertaken (i.e., center of mass and timing).

4. Conclusions

After assessing the results of the simulation of molecular dynamics, specifically 20K particles over 10 timesteps, these are the key findings:

- In the Serial code, which was the simplest of all code implementations, the optimizations significantly helped. However, the level of optimization to choose will depend on the tradeoff between the time to complete all time-steps and debuggability of the code.
- Considering the fact that the serial code was executed on a single core of the node (i.e., it did not utilize the other available cores of the machine), it is unable to compete with the implementation of the MPI code and OpenMP code, which were deployed on multiple cores of the same node. As a result, the MPI and OpenMP code effectively addressed the parallelizable workloads.
- Taking into account that the MPI library was designed for parallel programming, which assigns an MPI process to each logical core of a node, and considering that the provided MPI code had several sections that could be parallelized, it demonstrated significantly better performance than the Serial code, although not surpassing the performance of the OpenMP code (for instance: according to the data, MPI codes registered greater efficiency losses than the OpenMP codes).
- Despite the fact that OpenMP and MPI are functional for parallel programming, the existing physical constraint in this project (i.e., working on a single node) hurts the performance of the latter. As a result, the performance of the MPI code is degraded due to resource sharing (i.e., contentions associated with the current shared-memory situation), and anticipated overheads related to communication and synchronization. However, in a hypothetical situation of deploying the same simulation of molecular dynamics on a network of several nodes, the MPI code will surpass the performance of the Serial code and OpenMP code.
- The event of super-linear speedup puts into context how important the metrics of efficiency and speedup are in quantifying performance changes and scalability analysis when the problem to address or available resources remain fixed or increase.
- ***Finally, the OpenMP code has provided enough functional and numeric evidence to conclude that it is the most suitable code option for simulating the source code of molecular dynamics on a single node.***

5. References

Mayor of London (2021) *Record-breaking growth in London's cycle network continues*. London City Hall. [Online] https://www.london.gov.uk/press-releases/mayoral/mayor-and-tfl-announce-work-on-four-new-routes.

Ye, C. (2024) 'The current state of big data analytics research in marketing: A systematic review using TCCM approach.' *Journal of global scholars of marketing science*. Taylor & Francis, March, pp. 1–23.

Guzhov, S. and Krolin, A. (2018) *Use of big data technologies for the implementation of energy-saving measures and renewable energy sources in buildings*. IEEE Xplore. [Online] https://ieeexplore.ieee.org/abstract/document/8488861.

Alam, M. and Amjad, M. (2019) 'Weather forecasting using parallel and distributed analytics approaches on big data clouds.' *Journal of Statistics and Management Systems*, 22(4) pp. 791–799.

OpenMP Architecture Review Board. *omp_get_wtime* (n.d.) www.openmp.org. [Online] https://www.openmp.org/spec-html/5.0/openmpsu160.html.

Free Software Foundation, Inc. *Optimize Options (Using the GNU Compiler Collection (GCC))* (n.d.) gcc.gnu.org. [Online] https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html.

The Open MPI Project. *mpirun(1) man page (version 4.1.3)* (n.d.) www.open-mpi.org. [Online] https://www.open-mpi.org/doc/current/man1/mpirun.1.php.

Sasko Ristov, Prodan, R., Marjan Gusev and Skala, K. (2016) 'Superlinear Speedup in HPC Systems: why and when?' *Computer Science and Information Systems (FedCSIS), 2019 Federated Conference on*, October.

6. Appendix

Link to Google Drive (BD Assessment + HPC Assessment): https://stummuac-my.sharepoint.com/:f:/r/personal/23624012_stu_mmu_ac_uk/Documents/Documents/HPC%26BD/BD/Submit_BD_HPC?csf=1&web=1&e=lpJFv2