Department of Computer Science

UNIVERSITY of York

Submitted in part fulfilment for the degree of BSc.

Exploring Rust for Modelling Epidemics with Census Data

Samuel Ralph

May 2022

Supervisor: Steven Wright

Acknowledgements

I would like to thank my project supervisor Dr. Steven Wright for his assistance and support during this project. As well thanking my family and friends for supporting me throughout this project.

Contents

List of Figures

List of Tables

Abstract

The COVID-19 pandemic highlighted the importance pandemic simulators have on trying to contain the spread of disease by predicting the effectiveness of interventions. This study aims to produce a simulator capable of achieving this by utilising large datasets, like UK Census Tables and Mapping Data to provide a better model of the real world and improving the accuracy of the simulator. It also explores the feasibility of the programming language Rust, in high performance environments.

Executive Summary

Trying to accurately predict and model the spread of a disease through a population is an incredibly difficult task. But as shown throughout the COVID-19 pandemic, being able to accurately achieve this provides invaluable data that can help governments make informed decisions on how best to counter a disease.

Furthermore, technology is accelerating at rapid pace and what was considered impossible 10 years ago is freely available now [1]). This means that once state of the art simulators are now unable to take advantage of this new computing power and increase the fidelity of the simulations by providing even more data.

Therefore the aim of this project was to explore the impact that feeding huge amounts of data (including full maps of the UK, census data and more) can have on the accuracy and performance of an Agent Based simulator.

This will also be combined with a study into how a relative new language (Rust) can be used to speed the development process by using new memory management techniques in a high performance computing environment. As well as comparing the performance of this language against long term favourites (C/C++) in said simulation environments.

Investigation of previous work in producing simulators and using Rust in a high performance environment took place first to discover what had already been achieved, and also to create a set of minimum requirements that the success of the project can be gauged against.

Development of this project was done via an iterative agile workflow. First by creating a base core model and then expanding the model with more features to improve either performance or accuracy. This approach also directly provided the ability to measure the impact of different features on the accuracy and performance of the simulator at each stage.

The conclusion and analysis stage determined that nearly all of the original requirements were achieved, and provided evidence for Rust being a suitable candidate for high performance large scale projects due to the ease of development and memory safety.

Further this project it resulted in producing a simulator capable of simulating and predicting to a reasonable accuracy the spread of COVID-19 in York during a 6 month period from the 4/08/2020 to 14/03/2021, all within a 5 minute run time on a workstation grade computer.

Ethics were an import part of this project due to the high fidelity of data involved on real world people. This is because key characteristics of individuals are modelled like: age, gender, occupation, location, etc making it theoretically possible to identify individuals from the very small sample sizes involved. Therefore care was taken to ensure that data is kept confidential and individuals cannot be identified from the simulator.

1 Introduction

1.1 Problem Overview and Motivation

During the COVID-19 pandemic the use of computers to predict and model the spread of a disease and how the use of different interventions can slow the spread was proved invaluable in saving lives across the world. This is because it allows for less restrictive interventions to be deployed on a localised basis instead of the country wide lockdowns that were originally used at the start of the pandemic. However to achieve this the simulations needed to be fast and accurate to be of use in the rapidly changing conditions by officials who are enacting changes to try and slow the spread of the disease.

This highlighted the lack of available simulators as there was no significant need for them before. Especially ones capable of using the large amounts of data freely available and vast computing power to help improve the accuracy of these simulations.

Another key consideration was that most of the existing simulators are written in older well proved languages like C and C++. Illustrating a short-coming in taking advantage of the new developments in more efficient, modern languages like Go and Rust which have novel approaches to try and reduce the number of memory related bugs that plague large software projects. For example 70% of security vulnerabilities in Microsoft products are memory related [2], so trying to find approaches that prevent this are incredibly important. Especially in high performance and critical applications.

1.2 Aims and Objectives

This project aims to produce an accurate and performant epidemic simulator that is capable of using large amounts of data that is publicly available to improve the accuracy when compared to other simulators. It should also be highly configurable allowing it to be used for a large variety of diseases in a range of scenarios.

In addition this project also seeks to gain an understanding of how the programming language Rust fares in the development of a high performance program. With the key considerations being development ease and performance comparisons to other languages.

Hence the high level goals are: - Explore the previous work undertaken in the simulator world, and high performance programming with Rust - Design and implement a solution that is capable of meeting said goals, whilst conforming to industry best practices - Compare and Evaluate the performance and accuracy of the solution to real world data and competing state of the art simulators

1.3 Report Structure

The report is structured as follows:

Chapter 2 - Contains a literature review exploring the previous work conducted on Epidemic Simulators as well as simulators specific to Rust. Also provides details on the data that will be provided to the simulator

Chapter 3 - Uses the knowledge gained from the literature review, to create a list of requirements for this project to be successful and explains the measures taken to ensure an individuals privacy to not violate any ethics agreements.

Chapter 4 - Gives a broad overview of the design of the simulator and also explains in detail the more complex components. In combination with what techniques have been used to optimise the performance.

Chapter 5 - Evaluates the performance and accuracy of the simulator as compared to real world COVID-19 data

Chapter 6 - Concludes the report by analysing the results and reflecting on the original aims of the project. It also contains recommendations for how this project can be extended in the future.

2 Literature Review and Background

2.0.1 Introduction

To approach the project, several key decisions had to be made, requiring research into several topics to determine the best solutions for approaching an Epidemic Simulation problem. Furthermore, an analysis of pre-existing research must be undertaken, so results can be incorporated and not reproduced.

2.0.2 Types of System Modelling

The first decision was which simulation modelling techniques to use, with the main choices being: Agent Based Modelling (ABM) and Equation Based Modelling (EBM).

Equation Based Modelling

EBM attempts to model complex and dynamic systems through a series of equations. It was first proposed by Jay Forester in System Dynamics [3], and the equations can range in complexity from simple constants, to accounting for variability over time (ordinary differential equations), and even considering both time and space, through partial differential equations.

It has the benefits of being quick to compute, meaning much shorter run times and providing the ability to iterate through different parameters quickly, allowing selection of the optimum values to best match the system to be determined easily.

However, due to its simplicity it struggles to accurately model more complex characteristics of systems and does not account for stochastic (random) elements [4]. Furthermore, it requires a homogeneous population (individuals being the exact same), as well as each individual being equally likely to encounter any other individual; which is often not representative of complex systems, especially with social groups, as individuals tend to be heavily biased towards socialising with a small subset of the population.

Agent Based Modelling

ABM is a technique that applies a set of predictable rules at the micro level, to a series of individuals (agents) that can interact with each other and the environment, in an attempt to model complex systems and capture unpredictable patterns at the macro level [5].

It is the near complete opposite approach of EBM, as it is computationally expensive, but is much easier to construct, can easily model many characteristics of an individual with little cost, and can account for non-heterogeneous contact patterns. Furthermore, as it is a stochastic model, each run will be different, allowing probabilities of events to be captured.

Hunter et al. [6] compare the two aforementioned methods in a measles outbreak in Ireland, and they reach similar conclusions to those above. For example, they discovered that 200 runs of the equation model can take seconds, whereas the agent based model can take days. However, the agent based model provides more detailed information about the outbreak which can help inform decisions made by public health officials.

They do note that more work should be produced exploring the capability of Agent Based Models to incorporate “pushed” vaccines in hard hit areas and isolation. Techniques which my implementation should hope to achieve.

2.0.3 Existing Epidemic Simulators

ANTHRAX Bio Terrorism Modelling

In 2004, in the Morbidity and Mortality Weekly Report [7] a paper was released that proposes a modelling system for a bio terrorism anthrax attack in Norfolk Virginia.

They detail a model for dispersion and infection using the U.S. 2000 census for population density, and characteristics (age/sex). To estimate the infected citizens, they predict which cell a given citizen will be inside, at a specified time, based upon age classification (young/middle aged/elderly) and activity (school/work/recreation/home) on given days of the week. They then use Census Workflow data to model travelling between different counties, and inverse exponential driving distance (with different weights for different time categories) where no data is available.

This is a very useful approach for modelling geographic dispersion of Citizens and their schedules, but they do not consider using this approach for human to human transmission. Probably due to the fact that anthrax cannot be transmitted between humans, but it could be used to model other diseases/outbreaks that are.

Furthermore, they use a novel approach of modelling the probability of a Citizen seeking certain types of healthcare. Where they use physician billing records and prescriptions for each individual in a given zip code, to the healthcare facilities in the area. This is then collated to produce probabilities for each Citizen.

This technique would be incredibly useful to expand to epidemic simulation, as you could predict the probabilities of a Citizen detecting they are infected, and what measures they themselves would take to increase/decrease the spread of the disease.

FRED

In 2013, a state of the art agent-based Epidemic Simulator called FRED (A Framework for Reconstructing Epidemiological Dynamics) was produced. It was released in 2013, and utilises U.S synthetic populations from RTI [8] to model pandemics. It incorporates schools, households and workplaces, as well as modelling the impact that school closures and vaccines have on a pandemic. However, it does not account for public transports or shops, (which during COVID-19 we learned have a big impact on the spread of a epidemic). Furthermore, its time step is limited to one day, which does not account for more complicated schedules for agents.

As it was produced over a decade ago, computing power has significantly advanced since then [1], it would be interesting to see how the increase in computing resources can improve the accuracy and detail of an Epidemic Simulation; for example modelling public transport and shopping, with more frequent time steps, as well as incorporating new/alternate techniques to fight epidemics (lockdowns, partial shutdowns).

2.1 Simulating in Rust

2.1.1 Introduction to Rust

Rust is a relatively new multi-paradigm, general purpose programming language first conceived by Graydon Hoare in 2006, that is designed for performance and safety. It combines both Systems Programming and low level memory management, with higher level features like functional programming. It also utilises a novel approach to memory management using an ownership model, which can guarantee memory safety at compile time, and when combined with it's strict compile time checks, massively reduces the chances of unexpected behaviour.

The Mozilla Foundation got involved in 2009 to help develop the language [9]. The first major release was in 2015, and major corporations like Dropbox, Cloudflare and Discord are using it/moving their codebase to it [10] [11]. Rust has also been the language of choice to develop high performance agent based simulations as detailed below.

Rust AB

Authors et al. compared the performance and speed of Rust in agent based envirioment with pre existing simulators like MASON [12] [13]. They provide a parallel implementation boasting much faster performance and scalability compared to similar projects. This is useful to see that Rust has already been considered for Simulation projects, and a good example of a proof of concept.

However, they mainly focus on the wider field of Agent Based Modelling, not specifically Epidemic modelling. And they only offer a rudimentary framework, with poor documentation (no docs.rs page) and do not host on the standardised library hosting site for Rust crates.io, which makes it difficult to incorporate into other projects.

Furthermore, the comparison they use is a Boids model [14] to illustrate performance, which is drastically less computationally intensive than sophisticated Epidemic modelling, so there is no telling how suitable it is. Nor does it provide any utilities for incorporation of large data sets, like UK Census Tables.

EpiRust

In 2020 a team in India (thoughtworks) implemented an epidemic simulator in Rust [15]. It was an effective fast model capable of simulating millions (3 million) agents in a large scale city. It was also capable of implementing several countermeasures to diseases like vaccinations, lock downs and mask wearing. They attempted to increase the speed of the simulation by applying a distributed approach to allow the simulation of multiple cities at once. However this is completely independent and not capable of simulating agents moving between these cities. Furthermore, the movement simulation for agents is very rudimentary (split into several blocks), and not an accurate representation of the real world at all. Nor does it take into account different types of professions, and the different risks they face in epidemics (service industry vs office workers). Furthermore, they do not take into account the densities and attributes of a population in certain areas. Which is incredibly important for high accuracy simulations allowing for detection of potential "hot spots" where case rates are particularly high.

They do prove the usefulness of Rust, as even though the entire simulation is single threaded, it still had "reasonable" performance. Although being single threaded is a hindrance for scaling up the model to simulate larger populations. Therefore it would be useful to explore the multi threaded performance of Rust.

2.2 Data

2.2.1 Census Tables

Every 10 years the UK government conducts a survey of the entire population recording detailed information about demographics. This data is used to determine resource allocation for different areas, given the population size and classification. This data is publicly available for anyone to use, and is a great source for creating simulation of the UK population.

All the UK census tables can be sourced from the NOMIS API [16] which is highly configurable service that can be used to specify exactly what data to retrieve (like specific columns and filters), and the format that it is presented in e.g. XML, JSON, ... etc).

The CSV data format is used in this project, due to its efficiency in data size (not repeating column names for every single record), and the higher throughput available through NOMIS. Once the CSVs are downloaded, the records are parsed in a 2-step process, first building a representation of each record, then grouping records by output areas.

Note, that for simulating the entire UK in the final results, the tables had to be downloaded in bulk from the API, due to resource limits set by NOMIS and it taking too long to filter and organise the data. (~ 200 seconds per $1,000,000$ rows, with $175,434$[2] total rows). This required a slightly different approach in importing the data, due to differences in data structure.

The tables used for this project are listed in Table: 2.1, but in summary the full population count for key groups (i.e. gender) per Output Area is incredibly important for determining the number of Citizen types. Further, the Age structure table which lists the number of Citizens at each age (yearly) is a key data point that can be used to for additional data

2.2.2 Open Street Maps

Open Street Maps is a global collaborative project to create a free geographic database of the world. It is registered as a not for profit foundation in the UK and was created in 2004 by Steve Coast. It now has over 2 million registered volunteers, and is used by many large organisations including Microsoft, Snapshat, Uber and more [17], [18].

For this project the entire dataset for England was downloaded in a PBF format from Geofabrik. The protobuf file format was chosen due to the compression factor being far greater than XML, while also being faster to access **pb**. This could then be easily loaded and parsed using available libraries.

NOMIS Table Code	NOMIS API Code	Name	Description
ks101ew	NM_144_1	Usual Resident Population Per Area	This details number of citizens split by gender per Output Area.
ks608uk	NM_1518_1	Occupation Counts Per Area	This details the number of workers in each Occupation Type (Standard Occupational Classification 2010 (SOC2010)), per Output Area
wf01bew	NM_1228_1	Location of usual residence and place of work	This lists the number of Citizens commuting to every Output Area for work, from their residential Output Area.
ks102ew	NM_503_1	Age Structure	This details the number of Citizens per age (accurate to a year)

Table 2.1: NOMIS Tables

The dataset consists of millions of nodes, paths and relations, each with tags describing that feature. Nodes are a single position in space, ways are a collection of nodes joined together to form an outline, relations are used to group ways together that share a common relationship, i.e. a bus route. Tags are a key value data structure applied to any of the elements, to describe a purpose. For example, a restaurant will have a building tag, and a tag stating “amenity=resturant”.

2.2.3 Output Area Boundaries

The UK census data is classifieds at varying geographical levels and to ensure the highest accuracy, the lowest level available are used, which is Output Areas [19]. Despite the tiny size of an Output Area, they are required to have a minimum size of 40 resident household and 100 residential people to ensure confidentiality of the individuals residing in said area. This reduces the ethical concerns with modelling real world individuals as uniquely identifying characteristics are hidden in the volume of data.

The boundaries for this dataset were sourced from the UK geo portal in the format of a Shapefile [20]. Shapefiles were chosen due to the significant reduction in file size, faster performance, and popularity in open sourced libraries compared to other formats like Geo JSON.

3 Requirements

As the aims of this project are quite broad and nonspecific it was deemed worthwhile to break them down into smaller achievable steps. Especially given that this problem is very open ended, as there are always more features that can be added to the simulation to improve accuracy. Hence initial targets were declared from the outset, to ensure that a functional model was built under the time constraints, and the minimum requirements for the implementation to be a success are detailed in Table 3.1

3.1 Ethics

A big consideration for this project is that it is simulating and approximating real people in their daily lives, so measures must be taken to ensure an individuals privacy so that if a random Citizen is selected from the simulation, then it cannot be linked back to any real world human.

Several measures were taken to achieve this, firstly the source of the data itself undergoes it's own confidentiality measures including record swapping for minority classifications [21]. Furthermore in the simulator itself the synthetic population is generated from random sampling so that unique characteristics are blended together to not be representative of the real world.

For example, if there was 25 year old male doctor and a 50 year old female Teacher (in the real world), then the characteristics would be blended together to possibly produce a 50 year old male teacher and a 25 year old female doctor in the simulator. This will reduce the accuracy of the simulator, but it is a necessary measure to take to ensure an individuals privacy and confidentiality.

ID	Description	Priority	Justification
R1	Be able to easily configure and alter disease parameters	Shall	To be of use in future and with different strains of the disease, it must be easy to change and tweak the disease parameters
R2	To be able to accurately simulate any area of any size in the UK	Shall	To be of use in the real world, the simulator should be able to easily switch between different areas of different sizes for authorities to best predict the impact of a disease in a given area.
R3	The simulator should accurately model daily life movements of Citizens (home/work)	Shall	To provide a good representation of the real world modelling what people actually do in their daily lives is a must.
R4	The simulator should be fast enough to model long periods of time (months) within a reasonable computation time (hours)	Shall	To be able to compare the effect of different interventions in real time, then the results must be able to be collected within a reasonable timespan, otherwise the project is useless
R5	Performance Metrics should be captured	Shall	To be able to compare the simulator to others, and to locate bottlenecks metrics of how the simulator performs are essential to collect.
R6	Utilise UK census data and mapping data	Should	To have the best accuracy of the real world the simulator should make use of these datasets to best model the given areas with the specific characteristics of that population.
R7	Interventions should be available to predict the affect they have on a disease	Should	To be of use to officials trying to reduce the spread of a disease, being able to see in real time the effect different interventions would have is an incredibly important aspect.
R8	The simulator should utilise multi-threading to ensure scalability	Should	Multi core computers are a key part of the modern world, and to extract the most performance from the simulator, this is a requirement.

Table 3.1: Project Requirements

4 Method

An iterative process was used to develop the simulator, as it allows for easy comparison of the impact different features have on the accuracy and performance of the simulator. This meant that each iteration built upon the previous iteration, expanding the capabilities each time.

An agent based methodology was chosen as opposed to an equation based approach due to the ease of adding in new rules that agents have to follow; furthermore it can better model complex interactions that an equation based one can not consider, as discovered in Section 2.0.2.

4.1 Data Structure

The parent data structure for the Simulator during the simulation, is a Struct called Simulator which contains collections for OutputAreas and Citizens, with several additional parameters like the Disease Model. (See class diagrams for more detail).

Each abstracted object in the Simulator will also have a unique ID Struct, inheriting from a ID trait . This is so IDs can easily be passed around using the ID trait and so key information can be contained in one place, as well as providing a general type for lookups that is efficent in memory usage. To ensure uniqueness every ID will have a random UUID, which is near guaranteed to be completely unique [22]· Example objects being Buildings, Citizens, Output Areas, etc. The Citizen Struct is a abstracted representation of a Citizen, thus contains key details like: age, occupation, disease status and home/work building ID's.

Output Areas were originally used to just store the different buildings in a given area, as well as the polygon containing their shape which was used for geo locating and visualistion purposes. However, the global Citzen collection was later broken down per Output Area allowing for parallel execution of each Output Area per timestep.

Buildings are a generic trait, used to store all the Citizen ID's that visit said building. They later also had a real world location added and extra functionality for specific types of buildings. They also implement functionality to determine the Citizens exposed if a given Citizen is infected; allowing multiple areas to exist in a singular building (i.e. Classrooms in schools). The parameters for the disease and simulator are stored in a JSON serializable struct, allowing for easy export and tweaking.

4.2 Core Model

4.2.1 SEIR Model

The SEIR model consists of 4 stages: Susceptible, Exposed, Infected and Removed. Every Citizen starts at the Susceptible stage, and when they come into contact with an Infected individual they move to the Exposed stage, simulating the latency period in a disease before becoming transmissible. During the Infected stage an individual displays symptoms and infects other individuals that are susceptible. Finally the Citizen moves to the Removed stage, in which they can no longer contract the disease, due to a variety of reasons including being deceased, gaining immunity or vaccination.

The SEIR model was chosen as it provides the best abstract representation of COVID-19 (The main disease focused upon in this simulator), due to the latency before infected individuals start showing symptoms.

Citizens

The region of the UK chosen to simulate is broken down into Output Areas, containing on average 100-300 citizens, whom are randomly distributed to households evenly to match the population count in the census tables. Then each Citizen was assigned an Occupation so that the Occupation distribution for each output area matches the census tables. See Table 2.1 for more details.

Citizens with the same occupations are assigned to the same workplace buildings in the same Output Area, but randomly shuffled to reduce the chance of Citizens living and working in the same buildings. However with the introduction of public transport cross Output Area workplaces were introduced.

The number of workers per building was determined from the government employment densities guide (2nd edition)[23] which lists the number of employees per square metre for each occupation. Then all buildings were assigned a temporary fixed floor size of $1000m^2$ (until the introduction of OSM data), and the number of occupants was determined from the following equation: $num\_occupants = floor\_size * employee\_density$.

Each Citizen was assigned a schedule for their location during a standard day, which for simplicity was chosen to be the work building from the hours of 09:00 to 17:00, and the rest of the time they are at their residential buildings. Having a schedule allows for easy extension with new features such as public transports, lunch breaks, shopping, etc.

To complete initialisation of the simulation, a random output area was chosen for initial infections, and 10 random Citizens in that area were set to the infected status. Then for each time step of the simulation, each output area was iterated over, and for each Citizen, their schedule was updated, and current location was changed if the schedule required it. In addition, their “Disease Status” was updated and if the Citizen was in a contagious state (infected), they and their current building was added to a list of "exposures".

Once each Citizen had executed their schedule and updated their location, the list of exposures were applied, where for every Citizen sharing a building with an infected Citizen, they had $X\%$ chance of “catching” the disease and moving to the “exposed” status. A 2 step process to generate exposures was retroactively added as an optimisation because otherwise the same building could have 50 executions of the apply exposures function in a single time step, whereas with a batch method it only occurs once but with higher probabilities.

4.2.2 Simplistic Interventions

To accurately model the impact of real world measures to contain a epidemic, naïve attempts at lock downs, mask wearing and vaccinations were introduced. These measures were chosen as they were the most prominent during the COVID-19 outbreak, which was the largest scale pandemic in this generation’s history. Each intervention is triggered when reaching a certain percentage of the population is currently infected, and then removed once it drops below the threshold, as show in table A.2.

Mask Wearing

This is broken into two stages, on public transport, and in public areas, as this was fairly representative of the stages of the UK mask wearing policies. As there are many complex factors that affect the infection probability from wearing a mask that are out of scope for this project, a simplified approach was taken which just reduces the infection percentage chance when a Citizen is exposed to a infected individual. Despite it being rudimentary, the approach taken does make a noticeable difference to the number of cases.

In further iterations it might be worth considering the effectiveness of different masks, i.e. masks provided to essential workers are of a much higher quality than those given to the general population as evidenced in the COVID-19 pandemic. However this would be a complex feature to add with low overall benefit to the simulation so it is quite low priority. Another consideration would be the mask compliance ratios, as not every Citizen is fully compliant with wearing masks, or even adjusting the compliance ratios, with the number of infected cases.

For the public transport level intervention, the infection percentages were only reduced when the Citizen was on a public transport vehicle, whereas the Public areas reduce the percentage everywhere outside the Citizens residential location.

Vaccination

When the Vaccination process is triggered, at every time step, N individuals that are in the susceptible stage, are randomly chosen to have a vaccine administered, and they are moved to the Vaccinated stage in the SEIRV model. This vaccination stage, currently grants total immunity to the disease, so the Citizen can never be infected.

In later iterations it would be worthwhile to explore waning immunity in vaccines, multiple doses and a phased rollout plan to target the most vulnerable groups. The phased rollout is proven to to be the most effective and reducing mortality rates for COVID-19, and represented what the UK government did during the pandemic [24].

Ideally, the vaccination program will become user configurable, so the effectiveness of different rollouts can be compared. However, it does require a more complex model for agents, as underlying health conditions should be considered, as well as varying the effectiveness of the disease on different age groups.

Lockdown

A lockdown is currently quite simplistic, in that when the infection percentage threshold is reached, all Citizens are permanently sent home, and stay there until the infection percentage drops below the threshold. Citizens in the same household can still infect each other, but otherwise the spread of disease is contained. This is quite a naïve approach as lockdown does not guarantee that Citizens stay completely at home, nor does it account for the fact that Citizens still need to go shopping for food and essentials.

Furthermore, it does not allow for essential workers, which are a must in allowing a country to still function. So in future implementations, this should be added, as well as allowing for some mixing of households (via shops), or even considering individuals who refuse to comply with the lockdown [25].

4.2.3 Public Transport

To ensure that a Citizen's schedule is not simplistic, in that they instantaneously teleport between two buildings (work and home), public transport events are required. This occurs, by randomly selecting 20% of the population (at initialisation) to use public transport when travelling between work and home. Unfortunately, due to the fact that the precise locations of households and workplaces are unknown (Only the Output Area is known, which can be considerably large), public transport routes are between two distinct Output Areas. All Citizens travelling between two Output Areas are grouped into batches and use the same "bus". Then if anyone is infected on that bus, everyone else on the bus has an exposure event applied.

4.2.4 Open Street Map

Importing Data

Open street map (OSM) [17] data is used to allow buildings in the simulator to physically match a real world building. This provides a more effective population density map, as individuals are “living” where more houses are located (as well as accounting for bigger buildings, i.e. flats), as well as making other techniques more viable (like accurate public transport); because without that, it is very hard to calculate the direction and distance a Citizen needs to travel to get to work.

For this to work, the entire dataset for England was imported, and the grid coordinates converted to the one used by the OSM output area polygons. Then anything outside the specified region to simulate was discarded to reduce memory usage. Next, all the buildings were loaded into a format suitable for the simulator, and classified based upon the OSM tags.

For certain building classifications, one real world building/location can consist of many sub buildings with the same tag, for example a school. This means that the buildings are skewed to having 10’s of duplicate buildings in the same location. To resolve this any buildings within 100m of each other that have the same classification, are grouped together into one building (Households and workplaces are excluded from this).

Next each building was assigned to an Output Area. As this is incredibly computationally expensive (need to find what polygon contains a specific point, repeated millions of times), several techniques were trialled to optimise it.

Geo spacial Computing

As a large part of the simulator consists of trying to locate the closest points to a given point, and what polygon contains a given point, lots of research and experimentation was done to find the fastest and most efficient approaches.

Luckily lots of pre-existing research has been done on this, as it is a complex problem encountered in many scenarios. However, as Rust is a new language trying to find useful libraries that implement specialised algorithms and have good interoperability with libraries was a big problem.

There is a relatively standard library in Rust for geospatial primitives and simple algorithms (georust [26]). It contains datastructures for most shapes, Points, LineStrings and Polygons with simple algorithms like Intersection Checks, Area and Centroid Calculations, Rotations/Translations, etc. GeoRust also has great serialisation/deserialisation support for standard formats like GEOJSON, Shapefiles and PBF. Hence this was chosen as the foundation for all of the geospatial code for this project.

Polygons containing a given Point

To find the polygons closest to a given point a QuadTree [27] was used to store the bounding boxes of polygons. Then a rectangular region of any size (1x1, to the entire grid) could used as the input and any polygons inside that region would be returned.

Bounding boxes were used instead of the actual polygons as the "containing" check for a rectangle is considerably cheaper than for polygons (4 comparisons vs many), which adds up very quickly for the number of comparisons required in a QuadTree. The depth of a QuadTree is dynamic and is determined by ensuring the number of items in a Quad is below a specified limit to reduce the final index cost.

Once the bottom of the QuadTree has been reached, all matching entries are returned, optionally they can also be returned in an order closest to the center of the search area, determined using the Manhattan Distance.

This is useful because the number of items required depends on the calling function, i.e. several schools may need to be returned due to the limit to the number of teachers (cannot have more teachers than classes).

This is also the reason a Trapezium Map was not used, as it does not have the capability of returning the N closest polygons to a point. It would also be more complex to implement as there was not a library already in existence so in consideration of the work required for the rest of the project a slightly less efficient but more adaptable and simpler data structure was chosen.

Closest Points to a given Point

For Citizens to locate the nearest type of building to them (i.e Schools/shops) a very fast lookup function was required. The approach chosen was to build a Voronoi diagram [28] that uses polygons to store all the points that are closest to a given seed. During the implementation of this technique several methods were trialled to generate the Voronoi diagrams to find the most optimal.

Flood fill [28] was initially chosen due to simplicity but was quickly discarded due to how slow it was, and it's high memory consumption. It requires a 2D array for every single point in the grid, which becomes quickly unfeasible given the area to simulate. Fortunes LineSweep algorithm [29] was also trialed but was still too slow, and the library implementing did not have great interoptibility with anything else. Finally Delaunay triangles [28] were settled on due to their considerable speed compared to other approaches. Retrieval used the previous polygon lookup method.

These Voronoi diagrams were only built for building classifications that required the closest building to a Citizen: Schools and Shops. This is because they require a lot of computation to build, and for larger areas they require significant amounts of memory. For example, York has 7000 buildings classed as Schools, over a grid size of $100,000^2$.

Schools

Once the OSM data was imported it could be referenced to assign students to schools. Every Citizen of the age 16 and under was assigned the Student occupation as it is mandatory for UK children to be in full time education. Home schooling was ignored as it is such a small minority of the population, estimated to be less than 1% of the school population [30].

Citizens that are classed as students are assigned to the closest school (using the aforementioned point lookups) to their residential location. This is an overly simplistic approach, and not completely accurate as school catchment areas are rarely spherical.

Furthermore, it also does not account for different classifications of school (primary/secondary/etc), nor the different student population sizes they have (i.e. Secondary schools tend to be much larger than primary). The reason for this is that OSM does not provide any information on the classification of school, and it is a significantly non-trivial problem to lookup school details from coordinates.

Hence a comprise was achieved where although schools contain students from every age, students are grouped into age groups of a single year, so students only share classes with students their age.

Again, a naïve implementation was used for class assignment as one student was only assigned to one class, which is not representative of real schools where a student belongs to many classes with a large mixture of students.

This will have an impact on the spread of the disease, however this was done because it another complex problem to produce a full schedule with a mixture of classes for every single student. In future, with more time available this would be something worthwhile to explore, but given time constraints it could not be implemented.

Students are randomly shuffled and then assigned to a class per age group, so that the number of students per class is approximately 26 (the UK's average class size) [31].

Citizens with the teaching occupation, are assigned in a nearest neighbour fashion to the school closest to their residential location, that does not have enough teachers for each class. If a teacher fails to be assigned to over 20 schools, they are put to the side.

Once every school has reached capacity for teachers, the rest of the Citizens with the teaching occupation are assigned to the nearest school. However due to a quirk in the data, there are many more Citizens with the teaching occupation, than classes. So these extra teachers are assigned to an office job in the closest school.

This is not ideal, but it was the compromise that had to be made as the cause for the large number of Citizens with the teaching Occupation is unknown and the Occupation classifications do not have enough fidelity for other solutions.

Exposures and infections are similar to workplace buildings, in that any Citizen that shares the same class/office is infected. Potentially this could be improved by having a random distribution, so only the 5 closest Citizens are infected as it is unlikely an infected Citizen on one side of the classroom would infect everyone else. But this would add significant performance impact (needing to model the individual location/closeness of every Citizen as opposed to the grouping method currently employed.

4.3 Performance Improvements

As more features are added the time taken per simulation step was increasing significantly. Therefore it became worthwhile to explore the decrease in overall run time by leveraging different performance improvements. This had not been implemented previously as some of the techniques would cause significant code complexity that would slow down the rapid prototyping stage in adding new essential features.

4.3.1 Multithreading

A global thread pool, was created using the library Rayon[32] removing the need for manual management of a thread pool [33]. In addition due to Rust providing functional programming methods (in the form of Iterators), Rayon adds an easy extension to this with Parallel Iterators which are incredibly easy to use. Furthermore, data races are guaranteed not to exist as only one mutable reference can exist at any one time, which is enforced by the compiler. Rayon uses a work stealing implementation, which allows idle threads to take queued tasks from others and execute them [34].

However, to fully implement multi-threading the way data is stored in the Simulator must be altered, to prevent data races. Hence, the global Citizen HashMap was broken up, so each Output Area contains the Citizens currently in said area. This allows Output Areas to be executed in parallel, but after every time step, an individual Citizen may need to be transferred between two areas. Furthermore, a global lookup table is required, listing which Citizens are stored in which Output Areas, as they need to be retrieved from functions outside the Output Area they are contained in. See class diagram for detailed overview A.1.

4.3.2 Version 1.6 Vectors

One significant performance issue with the previous iterations is the use of HashMaps instead of direct memory access with vectors (extendable arrays in Rust). Although HashMaps are great for code simplicity and productivity, they require 2 memory lookups and an iteration of a Hash function. Whereas if you can store the indexes of an element then the memory lookups are halved, with no costly hash functions. This also aids cache utilisation as less data needed in the cache, so it can be used for other purposes.

To achieve this, all significant HashMaps (Citzens, Output Areas, Buildings, etc) are converted to Vectors. Then the ID structs for every element are modified to store the index of the array it is contained in. Also if it is in a 2D lookup, i.e. Citizens inside Output Areas, then the index of the outer array is also stored (the output area index). This means that any object can be accessed from anywhere in the main Simulator loop without having to use Hashmaps.

5 Results

5.1 Method

To compare the performance and accuracy of the different implementations/features a complete simulation was run for two different areas York and Yorkshire and the Humber for 5000 time steps). Two different areas were chosen as they provide a good comparison of how the density, number of Citizens and Output Areas affect the simulator.

The parameters provided to the simulator (Table 5.1) were altered to best approximate the sixth month period from September 2020 to February 2021.

Parameter Name	Value
Exposure Chance	0.00055
Exposure Time (Hours)	96
Infection Time (Hours)	336
Mask Adherence Percentage	80%
Mask Effectiveness Percentage	70%
Mask Public Transport Threshold	0.001
Mask Everywhere Threshold	0.0022
Lockdown Threshold	0.0034
Vaccination Threshold	0.005
Vaccinations Per Hour	80

Table 5.1: The values of the final parameters used

This period was chosen as it provides a range of real world interventions applied (lockdowns, masks and vaccinations) with two main peaks in cases. It is also probably one of the first times for accurate test reporting, as the first outbreak did not allow enough time for the test reporting process to be implemented.

Furthermore it also encapsulates the introduction of Vaccinations in the first few months of 2021 allowing the effectiveness of the vaccination intervention to be compared as later on most of the population had been vaccinated and new variants appeared with differing immunity making it much more difficult to compare.

	Workstation	Viking	CSGPU1
Processor	Ryzen 5 3600	Intel Xeon 6138	Intel Xeon 4114
Cores	6 @ 3.6GHz	@ 2.0 Ghz	10 @ 2.2 Ghz
Threads	12	40	40
Memory	32GB DDR4	192GB DDR4	192GB DDR4?

Table 5.2: Specification of Computers used

Three different computers are used for the simulations (Table 5.2) to investigate the impact that hardware has on the simulator and to explore how scalable the simulator is with more performant hardware. Reference data was retrieved from the UK Covid Dashboard [35] as it is a highly accurate source with high resolution for the entire UK covid-19 pandemic and it is available under the Open Government License [36].

A custom statistics logger was implemented to capture performance metrics and disease/infection metrics every time step. Total memory usage snapshots, total time taken and time taken for key individual functions are captured. High level global disease stats are recorded which is just the total number of Agents in each stage of the SEIR model. As well as the number of exposures that occurred in each Output Area/Public Transport location each timestep. All this data is dumped to separate JSON files for further analysis and comparison.

Compromises had to be taken in the amount of statistics captured, due to the sheer size the data generated could be. Hence although it could be useful to see exposures per building/agent it is infeasible to realistically analyse/report. Each data dump is organised based on the specific iteration and the parameters for each iteration to ensure fair comparison and good organisation. Jupyter notebooks were used to analyse the metrics and compute statistical results, as well as plotting graphs for comparison.

5.2 Real World Accuracy

The simulator results against the case numbers taken from the COVID dashboard are in figure 5.1. More examples are available in the appendix for different areas and population sizes but York provides a good example for the simulator.

5.2.1 Analysis

With the provided parameters the simulator matches the real world (and interventions) reasonably well. It predicts two significant waves and within margin predicts the sizes of the waves.

Unfortunately it does not completely match the slower decrease in cases after the first wave and also predicts the 2nd wave much sooner than

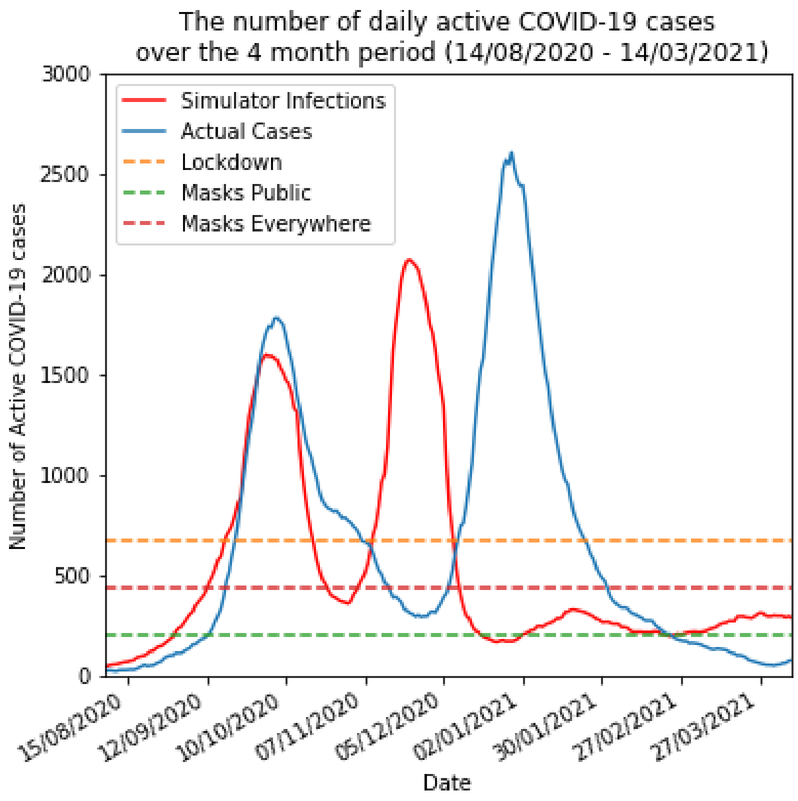

Figure 5.1: The simulator case numbers against the reference COVID dashboard case numbers

it actually occurs. However this is probably due to the coarse grained nature of the simulator lock downs and interventions compared to the real pandemic. i.e. the simulator takes a very binary approach in lockdowns whereas the UK government took a much softer approach in relaxing restrictions.

For example the regional 3 tier system introduced in October 2020 and again in November [37], allowed a increased freedom for Citizens whilst also still enforcing key measures.

Likewise the restrictions on social gathering (Rule of 6 [38]) helps to delay the peak in cases when not in a lockdown. But as the simulator predicted there would be another wave, this shows the usefulness of the simulator. With further work these variations on lockdowns and milder interventions could be introduced helping to improve the accuracy. Understandably it does not take account of the later peaks which were due to a more transmissible strain of the disease that is not modelled here.

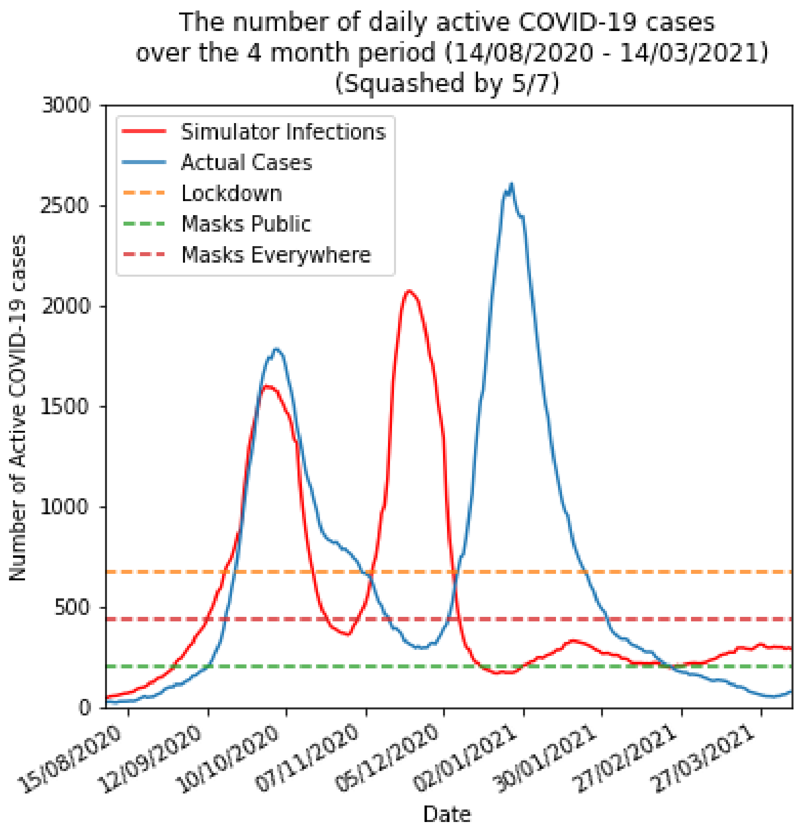

Figure 5.2: Shortened days

5.2.2 Weekends

One interesting thing to note is that the Simulator results seemed “squashed” when compared to real world results. i.e. The width of the peaks is somewhat less than the real world peaks. One possible cause for this is that the Simulator has a 7 day workweek whereas in real life, people tend to work 5 days. Also evidenced by the drop in case numbers on the weekends (Table: 5.3).

And as most of the transmissions take place on the way to work, or at work, this could mean that the real world cases occur at $\frac{5}{7}$ the rate of the simulator. To evaluate this, the statistics results were stretched horizontally be $\frac{5}{7}$ and plotted for comparison as shown in figure 5.2. This does provide a better match for the cases curve and in future the simulator should be adjusted to account for weekends.

5.3 Stochastic Nature

As each simulation is heavily stochastic as it depends on the exact starting infections and random decisions each Citizen makes during every single timestep the results are unlikely to be exactly repeatable. This is proven

Weekday	Mean Cases
Monday	94.27
Tuesday	88.26
Wednesday	90.02
Thursday	79.76
Friday	72.08
Saturday	60.03
Sunday	68.70

Table 5.3: The mean cases per day over the entire pandemic

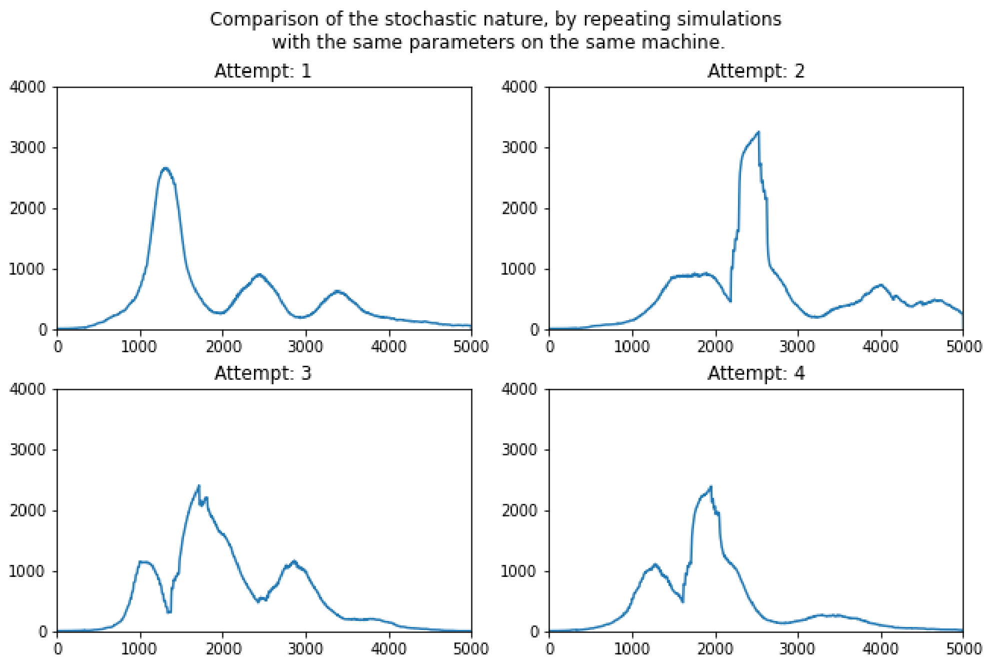

Figure 5.3: Variance in output with the same starting parameters

in Figure 5.3 where the a simulation was repeated several times with the exact same parameters.

This could potentially be solved by repeating simulations multiple times to gain an average approximation of how a simulation performs. This is unfortunately unattainable within the scope of this project as the compute requirements required for trying to run several repeated simulations for each variable that needs to be compared i.e. different iterations, cores allocated, machines, areas to simulate and starting parameters is far higher than what was available.

5.4 Performance

The Figure 5.4 shows a summary of the mean and std time taken for different optimisations for a simulation of York.

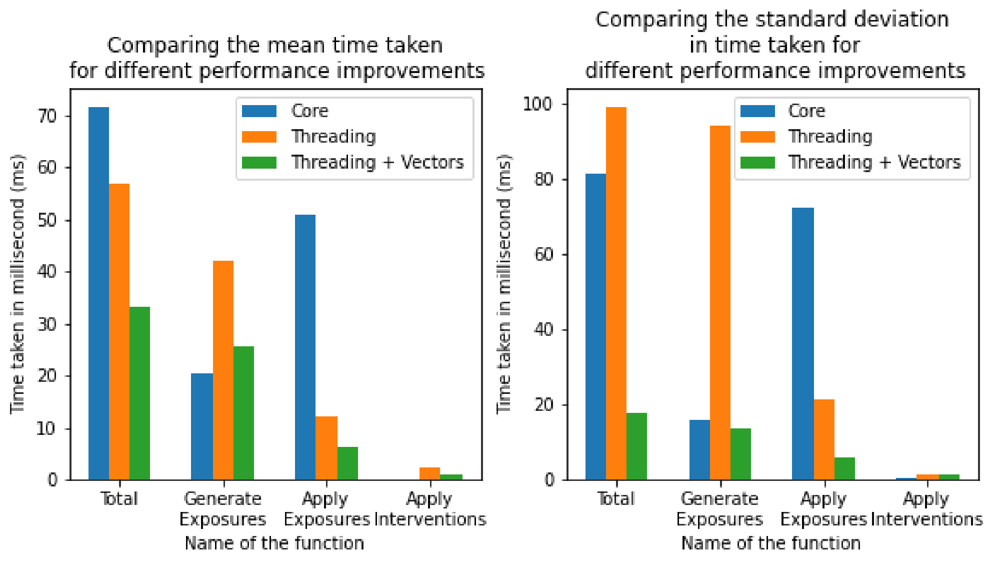

Figure 5.4: The average time in milliseconds for each function during a simulation of York on the workstation PC for a thousand time steps

There are large fluctuations in the time taken for the `generate_exposures` and `apply_exposures` function (as shown by the large standard deviation), and this is probably due to the non uniform distribution of exposures occurring. For example, the bulk of exposures occur during public transport and workplaces and less so at home. So there are spikes during those timesteps due to the sheer number of exposures and contacts Citizens are exposed to at that point.

This makes direct comparisons significantly more complex as the averages are unlikely to capture the full story of the time taken. Especially given each simulation being stochastic so results can differ even on the same machine with the same parameters.

Supplementary figures (A.4, A.5, A.6, A.7) are available in the appendix which show the time taken per function over the course of the entire simulation highlighting the fluctuations per time step. Note that Figure A.5 has had a savgol filter applied to smooth out some of the fluctuations.

Observations

As shown in figure 5.5 the importing of the OSM data, caused a massive spike in initialisation time due to the significant geo computation required, even with multithreading in key functions (assigning buildings to output ares). This meant that the threading optimisation actually increased as having to use thread safe data structures everywhere added a small cost which was not outweighed by converting the limited initialisation functions that weren't

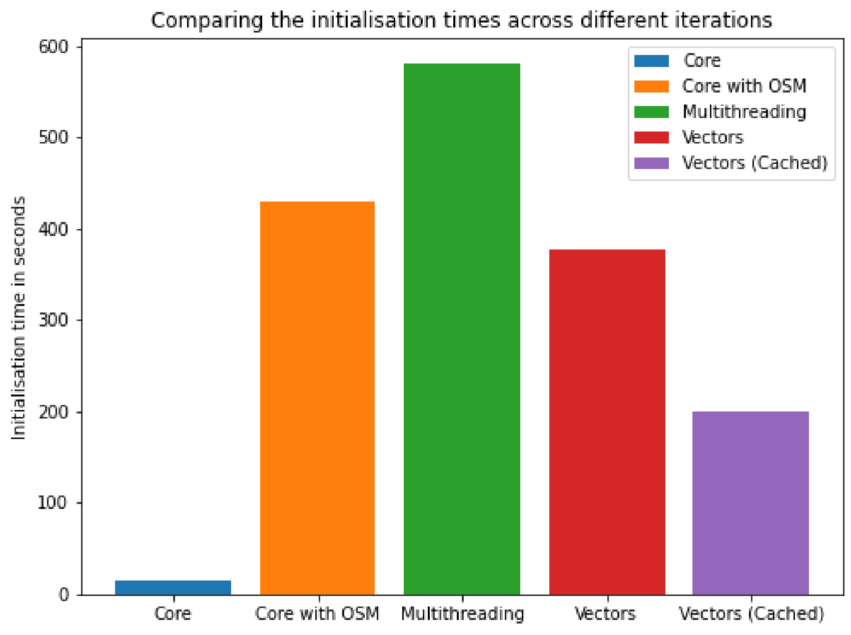

Figure 5.5: The initialisation times for different optimisations

previously multithreaded. However, even with the small additional cost for thread safe data structures the initialisation is such a small fraction of the total run time for the program that this issue is not a massive concern.

Furthermore the Vector optimisation caused a significant decrease in loading time due to faster lookups being available, so this already minor problem is completely removed, showing the benefit of Vectors instead of hashmaps in high lookup scenarios. This is also reinforced when using the “cached” feature which save a copy of the expensive map loading and conversion to disk for reuse.

Unsurprisingly the multi-threaded version was significantly faster than the core version with the peak time more smoothed out during the simulation, although the time taken per timestep fluctuates an incredible amount making it very hard to get an average approximation.

The Vector optimisation does not provide much benefit to the average time taken, but nearly eliminates all the fluctuations between timesteps making the simulator much more balanced and consistent in it’s performance. This could potentially be due to cache limitations as storing the additional lookup required for Hashmaps for large varied data exceeds the cache size available forcing slower fetches from main memory.

Memory usage did increase slightly but is still within margin of error and makes little impact given available memory given the machines it will be running on. i.e. $\pm 1/2$GB is very little difference on a machine with 200GB of RAM.

5.5 Scalability

Rather disappointingly the high core count servers do not actually yield much improvement over the workstation machine, but this could be down to the much higher IPC as the workstation has a much higher frequency and is also over 2 years newer. As well as the larger cache size (32MB vs 27.MB) and faster memory available.

As shown in the graphs, the memory usage increases as the area size increases, but not to a significant extent (doubling memory usage for a 18 fold increase in Citizens) showing that there should be no issue scaling to larger regions.

However, the run time scales not as favourably as it is more a $1 - 1.5$ ratio, so $18x$ more citizens, is a $26x$ longer runtime meaning that simulations take significantly longer to complete. Faster CPU's would reduce this further but has a significant cost to that.

Another approach would be implementing MPI allowing the simulator to scale horizontally removing the bottleneck of cores per machine on scaling the simulator, which would be worthwhile to explore in future iterations.

6 Conclusion

This report has explored the key components required in trying to create an Agent based simulator capable of accurately simulate a real world population and the spread of a pandemic throughout it. It started with a background study into the previous work undertaken in this area, and then used that information to design and produce a simulator. During production it was made clear that the original performance of the simulator was lacking and more optimisations were required to make it more competitive. Hence a restructuring of the key datastructures of the simulator was undertaken resulting in a significant boost to performance.

Then the simulator was evaluated and proven capable of accurately simulating the real life COVID-19 pandemic and predicting (with reasonable accuracy) the period from the 14/08/2020 to 14/03/2021 in York, England within a short space of time.

It has also proved to be more significantly faster and more accurate than competing simulators, like EpiRust [15] which can complete a simulation of 3 million Citizens in ~ 9000 seconds, as oppose to the ~ 4000 seconds it takes for this solution, a over $2x$ improvement. Whilst also being more accurate and representative of the real world. Admittedly EpiRust uses less memory, but that is a given due to the lack of real world mapping data and the detailed census data in the EpiRust model.

However, the scale of this project was significantly underestimated (over 9000 lines of code were written) as well as the complexity of using mapping data in the simulator, causing the bulk of the development time to be spent in trying to achieve a correct, high performance implementation. This meant that although the mapping data is now available in the simulator, it is not used to it's full extent (i.e. shops). But it is much easier now to develop from and add new features as the core constructs are already there. Similarly, this resulted in compromises to other areas like the user experience, so that the mapping data could be included.

However, given only one person was working on the simulator the amount that was achieved is significant in the time-span available and with a few improvements it could become a significant resource for pandemic modelling.

6.1 Evaluation against Original Requirements

As shown in table 6.1 all goals have been achieved, with the partial exception of R1, where command line configurable parameters are not enabled. But this is a trivial fix to implement and was not important to the actual development of the simulator. Apart from that all goals have been met to a high standard and in some cases exceeding the original expectations of this project.

ID	Status	Comment
R1	Partially	Although the parameters are stored in a JSON serializable struct, there is currently no way of altering them from the command line and they are only changed in the codebase.
R2	Complete	The simulator is able to easily scale up to a region the size of England (once the datatables have been downloaded).
R3	Complete	The simulator models people during a typical week with home, work/school and transport in between. In future it would be worth expanding upon this schedule.
R4	Complete	The simulator is capable of simulating a region of 3.5 million Citizens in approximately ~ 1.5 hours on a standard workstation computer.
R5	Complete	Time and memory usage per timestep is logged into statistics files.
R6	Complete	Census data and mapping data are used to great extent in the simulator.
R7	Complete	Lockdowns, masks and Vaccinations are able to be modelled.
R8	Complete	The simulator has been tested scaling to 40 threads.

Table 6.1: Evaluation against Original Requirements

6.2 Comments on developing with Rust

The strict compile time checks did cause frustration sometimes in trying to get some code to compile, they were worthwhile in the end as the number of errors occurring with unexpected behaviour was significantly reduced when compared to using other languages, and was almost a near guarantee that if it compiled it would work as expected. The checks also offered alternative solutions to the particular problem that were not originally considered, showcasing the full capability of the compiler.

However, the ecosystem for Rust is rather lacking especially for more niche scenarios where specialist libraries would be useful resulting in additional work to develop the simulator, i.e. specific geo spacial algorithms. Furthermore there is significant difficulty trying to profile and debug the simulator when executing remotely due to a lack of support from IDEs, which was one of the difficulties encountered in this project.

Another minor pain point would be the long compile times, as it is rather frustrating to wait sometimes several minutes to compile the program. Which is made even worse due to the number of compile attempts it can take to produce a build with no errors.

The Cargo system for building and adding dependencies is almost painless and a massive draw for using Rust due to it's simplicity and "just works" attitude. Finally as shown in the Results, the performance for the simulator is significant and does not suffer from any garbage collector issues unlike other languages [11].

Overall, it is definitely a worthwhile language to consider for large scale, high performance projects. As it will only improve over time as new libraries and features are added, and it offers many advantages from speed to a helpful developer experience.

6.3 Further Work

There are many areas of the simulator that would benefit from further development, as the scope of the project is never ending. However some key highlights would be expanding the complexity of a Citizens schedule to account for things like shopping, social events and weekends.

Another key consideration would be developing for support for something like MPI to allow the simulator to scale horizontally across multiple machines, offering a significant performance benefit.

Finally the user experience is definitely worth improving on, as it currently only offers a few command line options and any parameters affecting the spread of the disease are constants in the code, requiring the editing of code to change. This is definitely not ideal and not suitable for use by end users. Similarly all the report and graph generating is a manual process by modifying a jupyter notebook. So a more friendly way of editing parameters and generating reports would be advantageous for end users.

Bibliography

[1] G. E. Moore, *Cramming more components onto integrated circuits*, 1955.

[2] *Msrc - bluehatil - trends, challenge, and shifts in software vulnerability mitigation*, Accessed: 2022-05-06. [Online]. Available: https://github.com/microsoft/MSRC-Security-Research/blob/master/presentations/2019_02_BlueHatIL/2019_01%5C%20-%5C%20BlueHatIL%5C%20-%5C%20Trends%5C%2C%5C%20challenge%5C%2C%5C%20and%5C%20shifts%5C%20in%5C%20software%5C%20vulnerability%5C%20mitigation.pdf.

[3] J. W. Forrester, 'Industrial dynamics,' *Journal of the Operational Research Society*, vol. 48, no. 10, pp. 1037–1041, 1997.

[4] H. Van Dyke Parunak, R. Savit and R. L. Riolo, 'Agent-based modeling vs. equation-based modeling: A case study and users' guide,' in *Multi-Agent Systems and Agent-Based Simulation*, J. S. Sichman, R. Conte and N. Gilbert, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 10–25, ISBN: 978-3-540-49246-7.

[5] E. Bruch and J. Atwell, 'Agent-based models in empirical social research,' *Sociological Methods & Research*, vol. 44, no. 2, pp. 186–221, 2015, PMID: 25983351. DOI: 10.1177/0049124113506405. eprint: https://doi.org/10.1177/0049124113506405. [Online]. Available: https://doi.org/10.1177/0049124113506405.

[6] E. Hunter, B. Mac Namee and J. D. Kelleher, 'A comparison of agent-based models and equation based models for infectious disease epidemiology.,' in *AICS*, 2018, pp. 33–44.

[7] D. L. Buckeridge, H. Burkom, A. Moore, J. Pavlin, P. Cutchis and W. Hogan, 'Evaluation of syndromic surveillance systems — design of an epidemic simulation model,' *Morbidity and Mortality Weekly Report*, vol. 53, pp. 137–143, 2004, ISSN: 01492195, 1545861X. [Online]. Available: http://www.jstor.org/stable/23315704.

[8] *Rti u.s. synthetic household population™*, Accessed: 2022-01-14. [Online]. Available: https://www.rti.org/impact/rti-us-synthetic-household-population.

[9] *Rust faq*, Accessed: 2022-01-18. [Online]. Available: https://web.archive.org/web/20160609195720/https://www.rust-lang.org/faq.html#project.

[10] *Rust website*, Accessed: 2022-01-18. [Online]. Available: https://www.rust-lang.org/.

[11] *Why discord is switching from go to rust*, Accessed: 2022-01-18. [Online]. Available: https://discord.com/blog/why-discord-is-switching-from-go-to-rust.

[12] A. Antelmi, G. Cordasco, M. D'Auria, D. De Vinco, A. Negro and C. Spagnuolo, 'On evaluating rust as a programming language for the future of massive agent-based simulations,' in *Methods and Applications for Modeling and Simulation of Complex Systems*, G. Tan, A. Lehmann, Y. M. Teo and W. Cai, Eds., Singapore: Springer Singapore, 2019, pp. 15–28, ISBN: 978-981-15-1078-6.

[13] *Mason*, Accessed: 2021-11-23. [Online]. Available: https://cs.gmu.edu/~eclab/projects/mason/.

[14] C. W. Reynolds, 'Flocks, herds and schools: A distributed behavioral model,' ser. SIGGRAPH '87, New York, NY, USA: Association for Computing Machinery, 1987, pp. 25–34, ISBN: 0897912276. DOI: 10.1145/37401.37406. [Online]. Available: https://doi.org/10.1145/37401.37406.

[15] J. Kshirsagar, A. Dewan and H. Hayatnagarkar, 'Epirust: Towards a framework for large-scale agent-based epidemiological simulations using rust language,' Sep. 2020. DOI: 10.3384/ecp20176475.

[16] *Nomis api*, Accessed: 2022-03-06. [Online]. Available: https://www.nomisweb.co.uk/api/v01/help.

[17] *Open street maps*, Accessed: 2022-03-21. [Online]. Available: https://www.openstreetmap.org/.

[18] *Osm foundation*, Accessed: 2022-03-21. [Online]. Available: https://blog.osmfoundation.org/about/.

[19] *Ons output area guide*, Accessed: 2022-03-21. [Online]. Available: https://webarchive.nationalarchives.gov.uk/ukgwa/20160107193025/https://www.ons.gov.uk/ons/guide-method/geography/beginner-s-guide/census/output-area--oas-/index.html.

[20] *Shapefile technical description*, Accessed: 2022-03-21. [Online]. Available: https://www.esri.com/content/dam/esrisites/sitecore-archive/Files/Pdfs/library/whitepapers/pdfs/shapefile.pdf.

[21] *Uk census 2011 statistical disclosure control policies*, Accessed: 2022-05-04. [Online]. Available: https://www.ons.gov.uk/file?uri=/census/2011census/howourcensusworks/howwetookthe2011census/howweplannedfordatadelivery/protectingconfidentialitywithstatisticaldisclosurecontrol/statistical-disclosure-control-for-the-2011-uk-census_tcm77-189747.pdf.

[22] P. J. Leach, R. Salz and M. H. Mealling, *A Universally Unique IDentifier (UUID) URN Namespace*, RFC 4122, Jul. 2005. DOI: 10.17487/RFC4122. [Online]. Available: https://www.rfc-editor.org/info/rfc4122.

[23] *Employment density guide 2nd edition*, Accessed: 2022-03-21. [Online]. Available: https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/378203/employ-den.pdf.

[24] *Uk covid vaccine rollout plan*, Accessed: 2022-03-27. [Online]. Available: https://www.gov.uk/government/publications/priority-groups-for-coronavirus-covid-19-vaccination-advice-from-the-jcvi-30-december-2020/joint-committee-on-vaccination-and-immunisation-advice-on-priority-groups-for-covid-19-vaccination-30-december-2020.

[25] *Coronavirus and compliance with government guidance*, Accessed: 2022-03-27. [Online]. Available: https://www.ons.gov.uk/peoplepopulationandcommunity/healthandsocialcare/conditionsanddiseases/bulletins/%20coronavirusandcompliancewithgovernmentguidanceuk/april2021.

[26] *Georust*, Accessed: 2022-04-14. [Online]. Available: https://georust.org/.

[27] R. Finkel and J. Bentley, 'Quad trees: A data structure for retrieval on composite keys.,' *Acta Inf.*, vol. 4, pp. 1–9, Mar. 1974. DOI: 10.1007/BF00288933.

[28] F. P. Preparata and M. I. Shamos, *Computational geometry: an introduction*. Springer Science & Business Media, 2012.

[29] S. Fortune, 'A sweepline algorithm for voronoi diagrams,' in *Proceedings of the Second Annual Symposium on Computational Geometry*, ser. SCG '86, Yorktown Heights, New York, USA: Association for Computing Machinery, 1986, pp. 313–322, ISBN: 0897911946. DOI: 10.1145/10515.10549. [Online]. Available: https://doi.org/10.1145/10515.10549.

[30] *Adcs elective home education survey 2021*, Accessed: 2022-03-31. [Online]. Available: https://adcs.org.uk/education/article/elective-home-education-survey-report-2021.

[31] *School pupils and their characteristics*, Accessed: 2022-04-18. [Online]. Available: https://explore-education-statistics.service.gov.uk/find-statistics/school-pupils-and-their-characteristics.

[32] *Rayon github*, Accessed: 2022-03-03. [Online]. Available: https://github.com/rayon-rs/rayon.

[33] *Introduction to rayon*, Accessed: 2022-03-03. [Online]. Available: https://smallcultfollowing.com/babysteps/blog/2015/12/18/rayon-data-parallelism-in-rust/.

[34] R. D. Blumofe and C. E. Leiserson, 'Scheduling multithreaded computations by work stealing,' *J. ACM*, vol. 46, no. 5, pp. 720–748, Sep. 1999, ISSN: 0004-5411. DOI: 10.1145/324133.324234. [Online]. Available: https://doi.org/10.1145/324133.324234.

[35] *Uk covid dashboard*, Accessed: 2022-04-22. [Online]. Available: https://coronavirus.data.gov.uk/.

[36] *Uk open government license v3*, Accessed: 2022-04-22. [Online]. Available: https://www.nationalarchives.gov.uk/doc/open-government-licence/version/3/.

[37] *Returning to a regional tiered approach*, Accessed: 2022-04-26. [Online]. Available: https://www.gov.uk/government/speeches/returning-to-a-regional-tiered-approach.

[38] *Rule of six comes into effect to tackle coronavirus*, Accessed: 2022-04-26. [Online]. Available: https://www.gov.uk/government/news/rule-of-six-comes-into-effect-to-tackle-coronavirus.

A Appendix

A.1 Tables

Module Name	Module Description
load_census_data	Downloads, parses and imports census tables into a usable format
osm_data	Imports and loads the Open Street Maps data, provides conversions between Geographical coordinates, fast lookups and groups buildings into Output Areas.
simulator	Handles everything to starting and running the actual simulator
visualisation	Series of tools for graphing and understanding the results of the simulator

Table A.1: The different modules in the Simulator

Intervention Type	Infection Percent Threshold
Mask Wearing On Public Transport	1%
Mask Wearing Everywhere	2.2%
Lockdown	3.4%
Vaccination	5%

Table A.2: Infection thresholds for Interventions

A.2 Class Diagrams

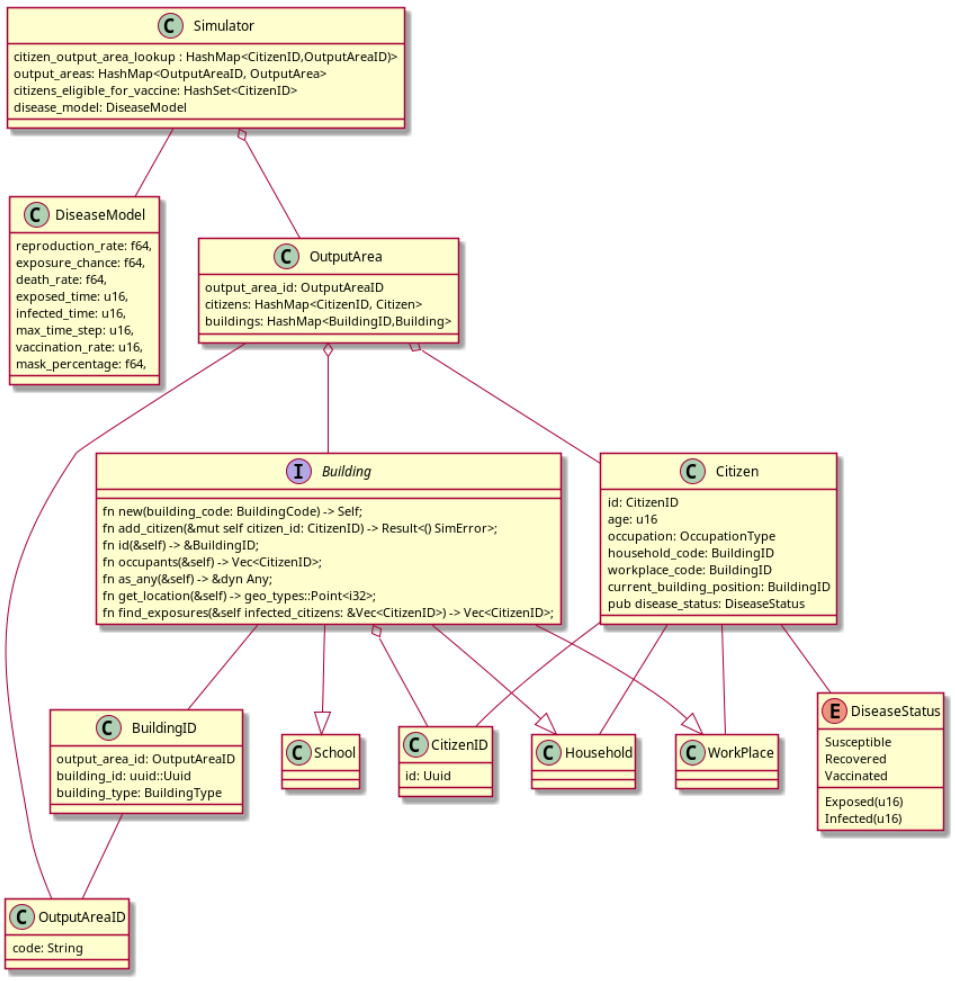
Simulator
citizen_output_area_lookup : HashMap<CitizenID,OutputAreaID)>
output_areas: HashMap<OutputAreaID, OutputArea>
citizens_eligible_for_vaccine: HashSet<CitizenID>
disease_model: DiseaseModel
DiseaseModel
reproduction_rate: f64,
exposure_chance: f64,
death_rate: f64,
exposed_time: u16,
infected_time: u16,
max_time_step: u16,
vaccination_rate: u16,
mask_percentage: f64,
OutputArea
output_area_id: OutputAreaID
citizens: HashMap<CitizenID, Citizen>
buildings: HashMap<BuildingID,Building>
Building
fn new(building_code: BuildingCode) -> Self;
fn add_citizen(&mut self citizen_id: CitizenID) -> Result<() SimError>;
fn id(&self) -> &BuildingID;
fn occupants(&self) -> Vec<CitizenID>;
fn as_any(&self) -> &dyn Any;
fn get_location(&self) -> geo_types::Point<i32>;
fn find_exposures(&self infected_citizens: &Vec<CitizenID>) -> Vec<CitizenID>;
Citizen
id: CitizenID
age: u16
occupation: OccupationType
household_code: BuildingID
workplace_code: BuildingID
current_building_position: BuildingID
pub disease_status: DiseaseStatus
BuildingID
output_area_id: OutputAreaID
building_id: uuid::Uuid
building_type: BuildingType
School
CitizenID
id: Uuid
Household
WorkPlace
DiseaseStatus
Susceptible
Recovered
Vaccinated
Exposed(u16)
Infected(u16)
OutputAreaID
code: String

Figure A.1: Abstracted Class Diagram for V1.5

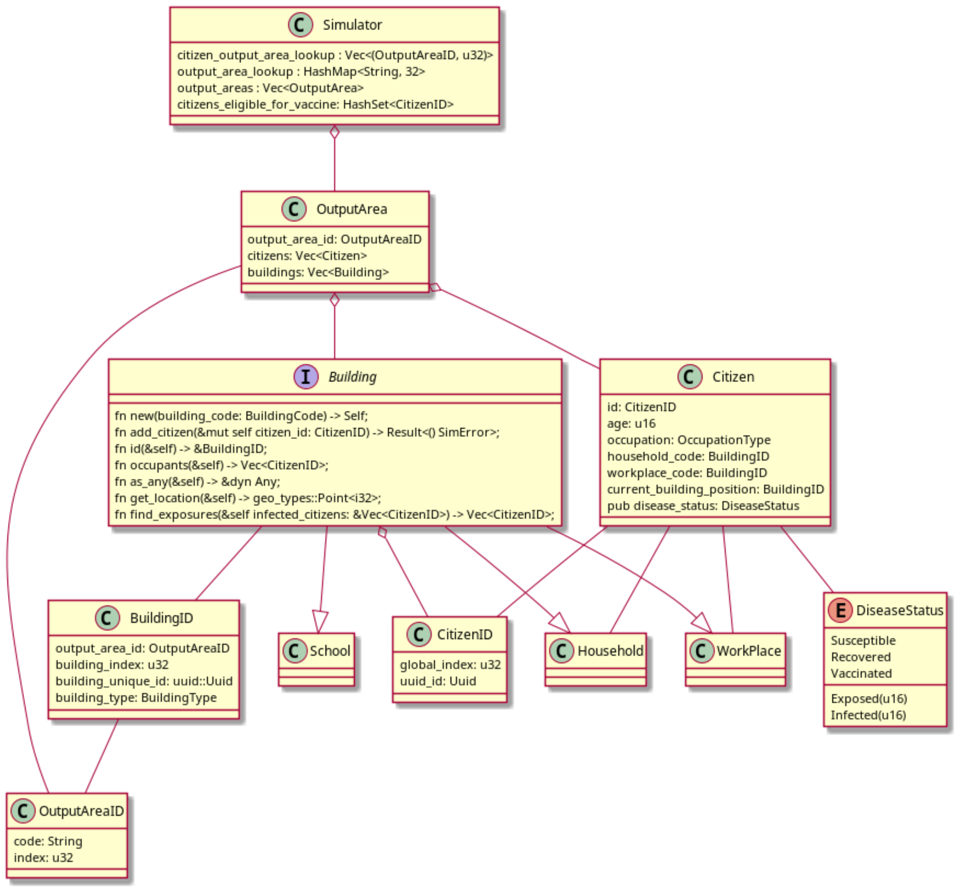
Simulator
citizen_output_area_lookup : Vec<(OutputAreaID, u32)>
output_area_lookup : HashMap<String, 32>
output_areas : Vec<OutputArea>
citizens_eligible_for_vaccine: HashSet<CitizenID>
OutputArea
output_area_id: OutputAreaID
citizens: Vec<Citizen>
buildings: Vec<Building>
Building
fn new(building_code: BuildingCode) -> Self;
fn add_citizen(&mut self citizen_id: CitizenID) -> Result<() SimError>;
fn id(&self) -> &BuildingID;
fn occupants(&self) -> Vec<CitizenID>;
fn as_any(&self) -> &dyn Any;
fn get_location(&self) -> geo_types::Point<i32>;
fn find_exposures(&self infected_citizens: &Vec<CitizenID>) -> Vec<CitizenID>;
Citizen
id: CitizenID
age: u16
occupation: OccupationType
household_code: BuildingID
workplace_code: BuildingID
current_building_position: BuildingID
pub disease_status: DiseaseStatus
BuildingID
output_area_id: OutputAreaID
building_index: u32
building_unique_id: uuid::Uuid
building_type: BuildingType
School
CitizenID
global_index: u32
uuid_id: Uuid
Household
WorkPlace
DiseaseStatus
Susceptible
Recovered
Vaccinated
Exposed(u16)
Infected(u16)
OutputAreaID
code: String
index: u32

Figure A.2: Abstracted Class Diagram for V1.6

A.3 Results Graphs

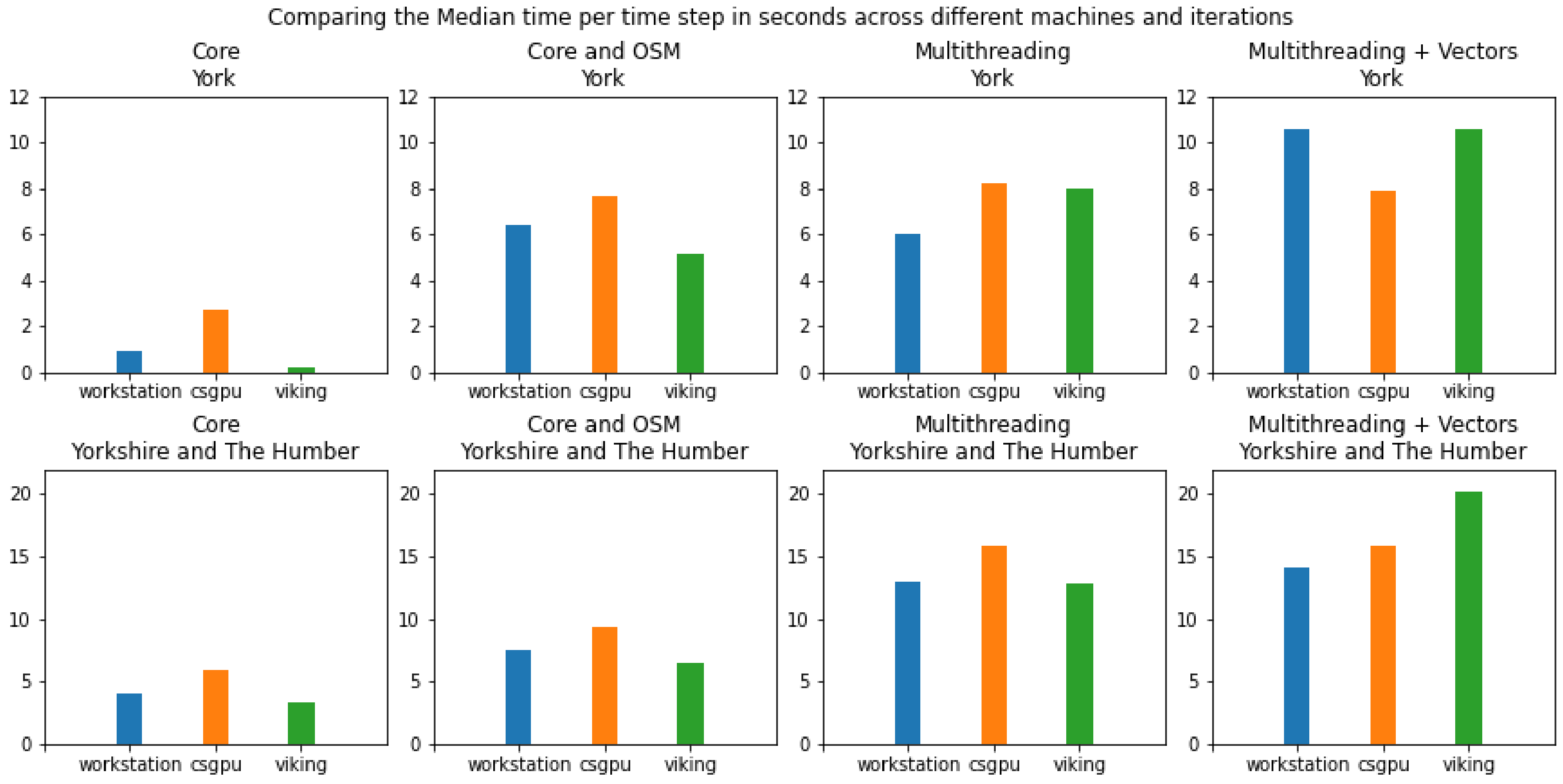

Figure A.3: Caption

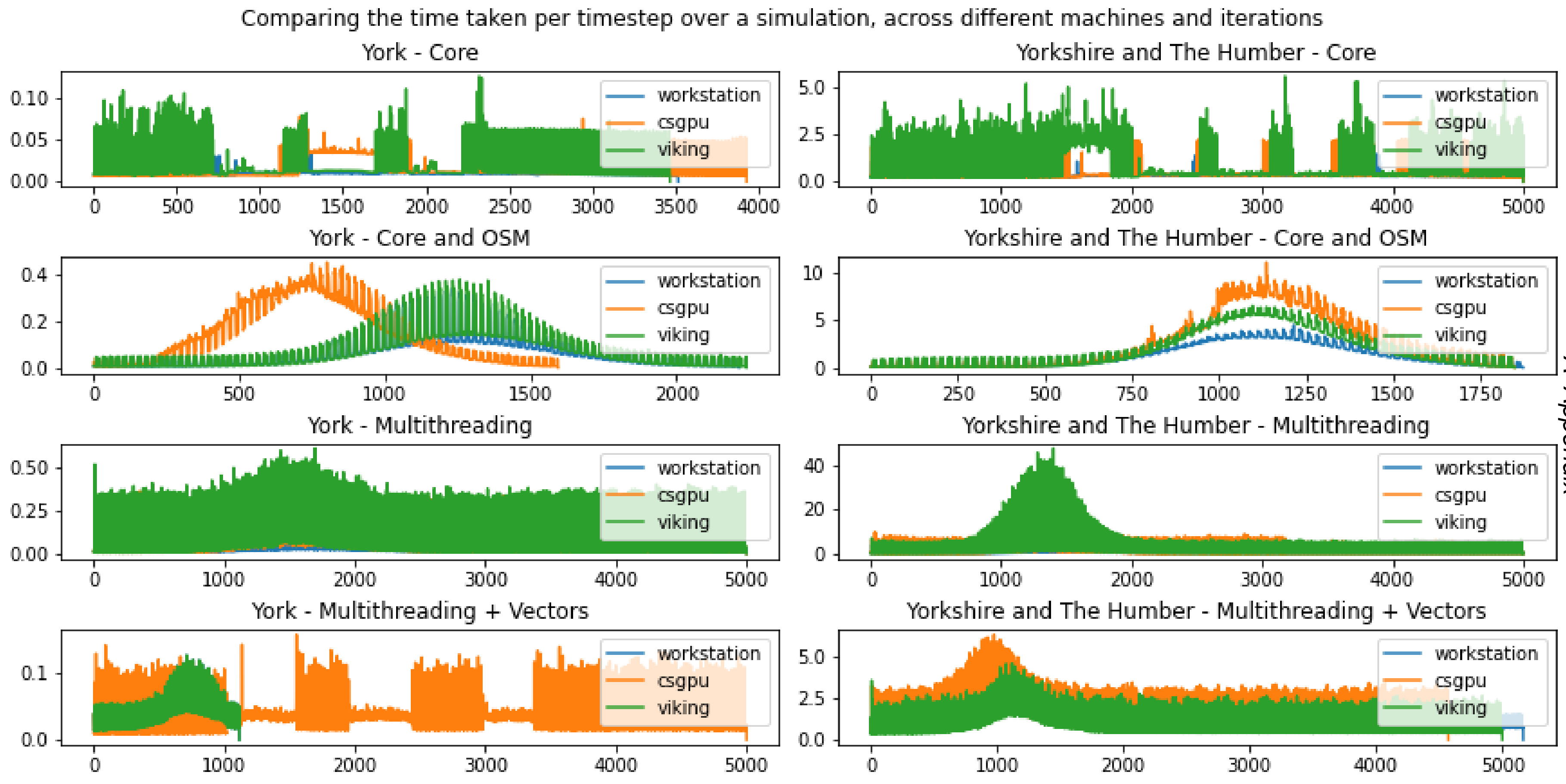

Figure A.4: Time taken per timestep for York

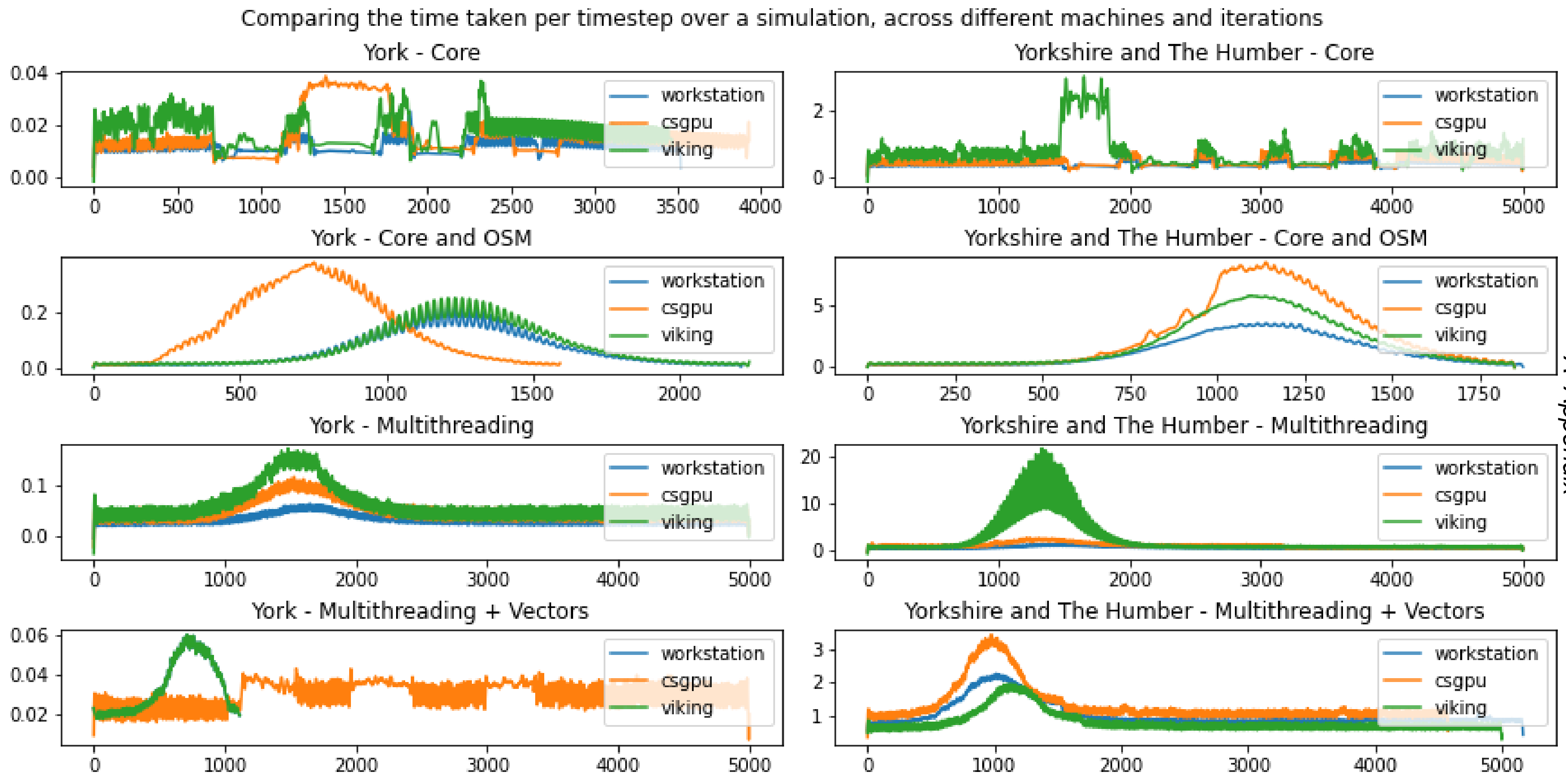

Figure A.5: Time taken per timestep for York

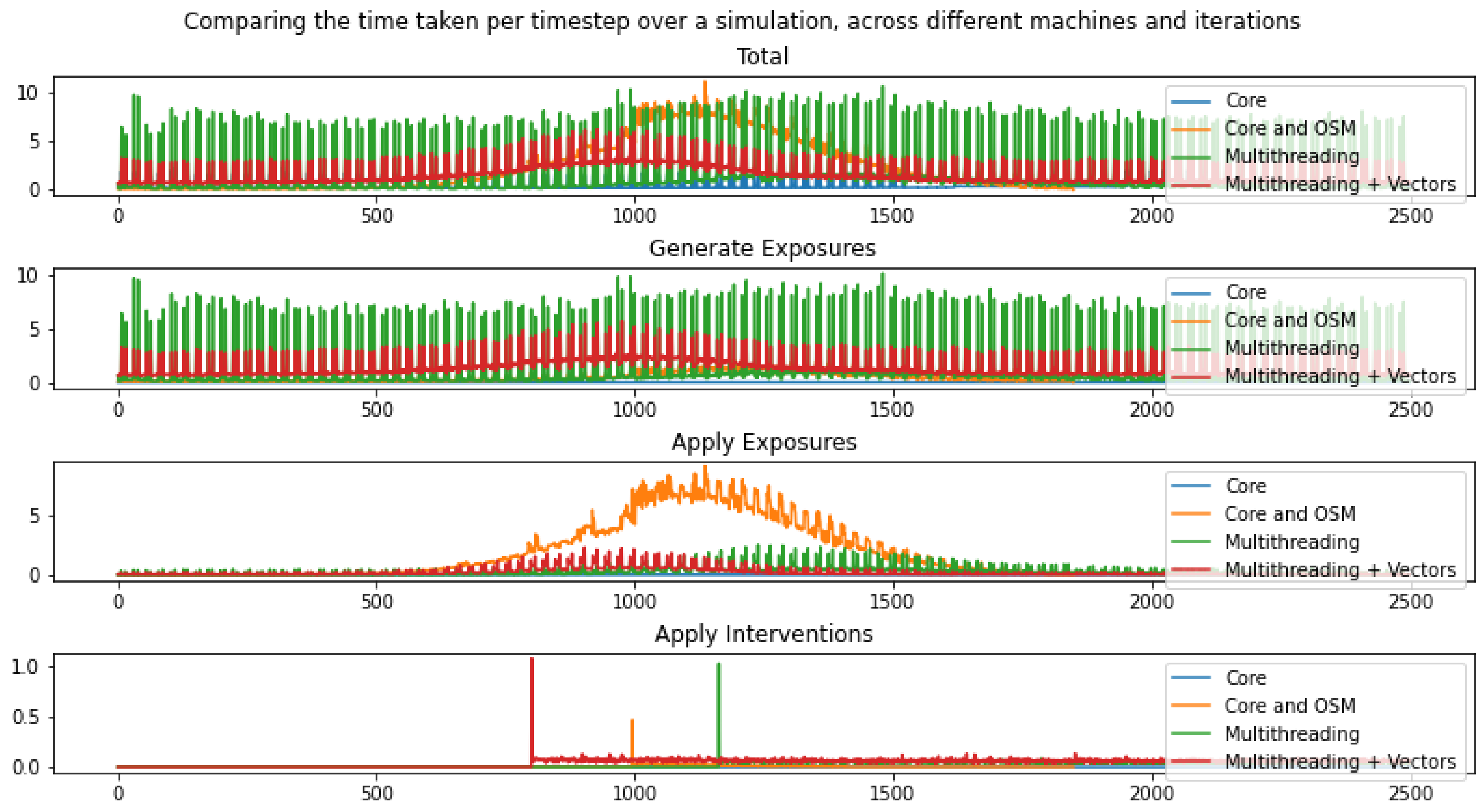

Figure A.6: The function time per timestep over the course of the simulation

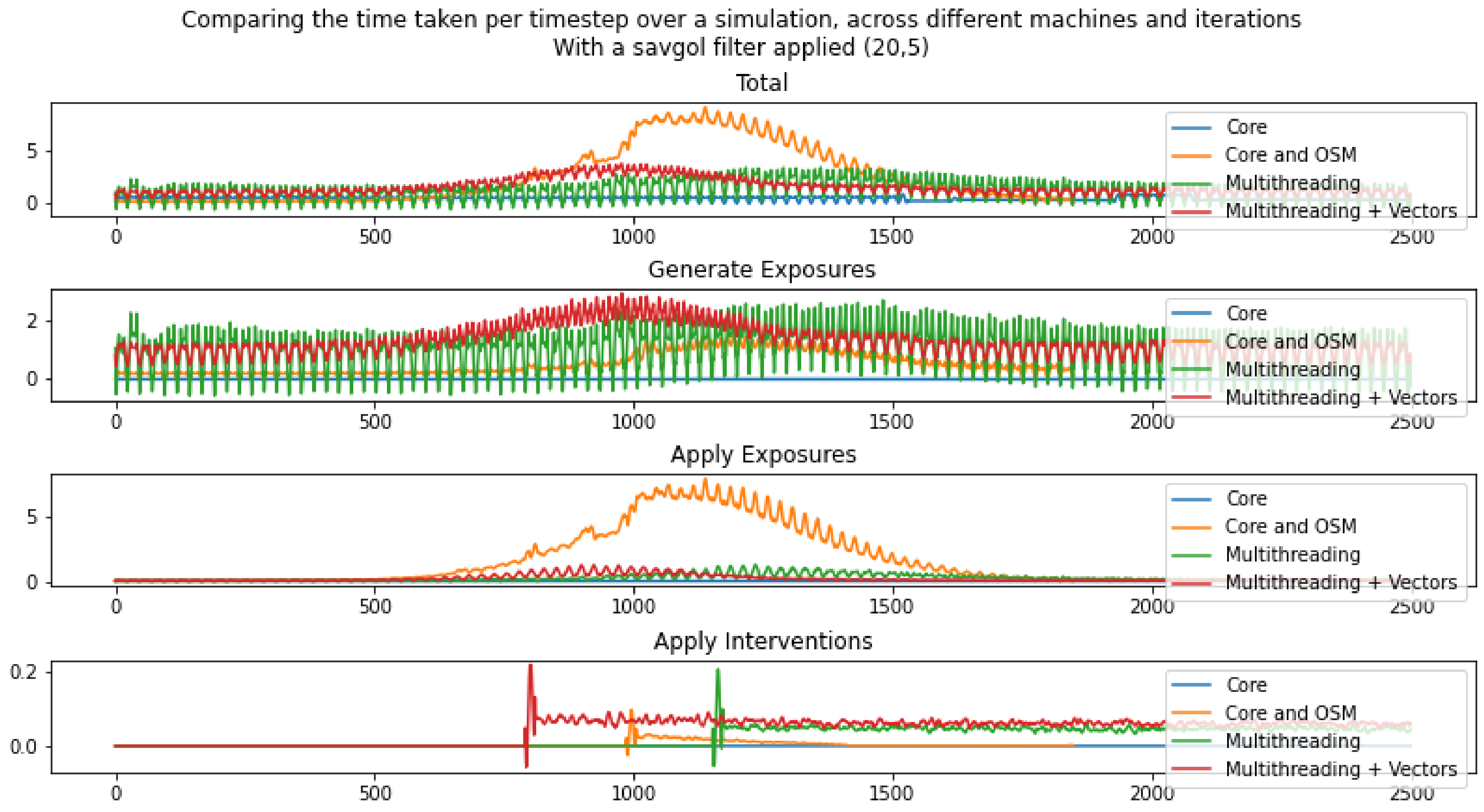

Figure A.7: The function time per timestep over the course of the simulation with a 20,5 savgol filter applied