# DOMAIN PARTITIONING AND SOFTWARE MODIFICATIONS TOWARDS THE PARALLELISATION OF THE buildingEXODUS EVACUATION SOFTWARE

**BIBI YASMINA YASHANAZ MOHEDEEN**

A thesis submitted in partial fulfilment of the requirements of the University of Greenwich for the Degree of Doctor of Philosophy

**September 2011**

# DECLARATION

I certify that this work has not been accepted in substance for any degree, and is not concurrently submitted for any degree other than that of Doctor of Philosophy (PhD) being studied at the University of Greenwich. I also declare that this work is the result of my own investigations except where otherwise identified by references and that I have not plagiarised the work of others.

X_____

Yasmina Mohedeen

X_____

Dr. Angus Grandison (first supervisor)

X_____

Prof. Edwin Galea (second supervisor)

# ACKNOWLEDGEMENT

I would like to thank Prof. Ed Galea for giving me every opportunity to fulfil my PhD. An exceptional mention goes to my brilliant supervisor, Dr. Angus Grandison. I cannot express my gratitude for his valuable advice, guidance and endless patience, without which I could not have succeeded in this long journey. I also would like to thank Dr. Peter Lawrence and Dr. Steve Gwynne for taking their time to review this dissertation.

My parents provided a lifetime of motivation and opportunity for me to pursue my ambition, for which I will always be indebted. I am also grateful to the people who have supported and believed in me during this research: my family, the Muthu family, the Cavanagh family, Lauren, Françoise, my wonderful friends and, last but not least, my FSEG family. Finally, I would like to offer my gratitude and thanks to Matthew Cavanagh for his precious support, encouragement and understanding, especially in the last hurdle of this journey.

For the precious bonding at our numerous tea breaks and badminton sessions; Steven's constant support and endless bickering since the beginning of this journey; Yasmine's precious advice, the amusing confusions that our similar names created and for giving me a good run for my money at being the clumsiest; Claudia, for the infinite patience and politeness for putting up with my distractions; Aoife, for the valuable literacy advice, for all the interesting views that our opposite selves brought and for all the kind 'pick-me-up' sticky notes I would find on my desk; Nitish, for all the crazy laughs at serious moments and our eternal plan to go to punting in Cambridge; BoumBoum, who believed in me more than I deserve and gave me the confidence to stand up tall – well as tall as 5ft can allow! One word: Priceless!

# ABSTRACT

This thesis presents a parallel approach to evacuation modelling in order to aid real-time, large-scale procedure development. An extensive investigation into which partitioning strategy to employ with the parallel version of the software was researched so as to maximise its performance.

The use of evacuation modelling is well established as part of building design to ensure buildings meet performance based safety and comfort criteria (such as the placements of windows or stairs so as to ease people's comfort) . A novel approach to using evacuation modelling is during live evacuations from various disasters. Disasters may be fast developing in large areas and incident commanders can use the model to plan safe escape routes to avoid danger areas. For this type of usage, very fast results must be obtainable in order for the incident commanders to optimise the evacuation plan along with the software's capability to simulate large-scale evacuation scenarios.

buildingEXODUS provides very fast results for small-scale cases but struggles to give quick results for large-scale simulations. In addition, the loading up of large-scale cases are dependent on the specifications of the processor used thus making the problem case unscalable. A solution to address these shortcomings is the use of parallel computing. Large-scale cases can be partitioned and run by a network of processors, thus reducing the running time of the simulations as well as the ability to represent a large geometry by loading parts of the domain on each processor. This scheme was attempted and buildingEXODUS was successfully parallelised to cope with large-scale evacuation simulations.

Various partitioning methods were attempted and due to the stochastic nature of every evacuation scenario, no definite partitioning strategy could be found. The efficiency values ranged from 230% (with both cores being used from 10 dual-core processors) when an idealised case was run to 23% for another test case. The results obtained were highly dependent on the test case's geometry, the scenario being applied, whether all the cores are being used in case of multi-cores processors, as well as the partitioning method used.

However, the use of any partitioning method will produce an improvement from running the case in serial. On the other hand, the speedups obtained were not scalable to warrant the adoption of any particular partitioning method. The dominant criteria inhibiting the parallel system was processor idleness or overload rather than communication costs, thus degrading the performance of the parallel system. Hence an intelligent partition strategy was devised, which dynamically assesses the current situation of the parallel system and repartitions the problem accordingly to prevent processor idleness and overloading. A dynamic load reallocation method was implemented within the parallelised buildingEXODUS to cater for any degradation of the parallel system. At its best, the dynamic reallocation strategy produced an efficiency value of 93.55% and a value of 36.81% at its worse.

As a direct comparison to the static partitioning strategy, an improvement was observed in most cases run. A maximum improvement of 96.48% was achieved from using the dynamic reallocation strategy compared to using a static partitioning approach. Hence the parallelisation of the buildingEXODUS evacuation software was successfully implemented with most cases achieving encouraging speedup values when a dynamic repartitioning strategy was employed.

# TABLE OF CONTENTS

# TABLE OF FIGURES

# TABLE OF TABLES

# TABLE OF TEST CASES

# TABLE OF EQUATIONS

# 1 INTRODUCTION

Large-scale evacuation is the mass movement of thousands or even millions of people [1] from a dangerous place, due to the threat or occurrence of a disastrous event, to a place of safety. The geometry involved is normally a very large area that can hold a large population such as stadiums, towns, high-rise buildings, etc. Very large events such as sports, religious, cultural and entertainment events are now regular occurrences. Organisers of these events have the difficult tasks of planning the evacuation from such places if any need arises. In recent years both man-made and natural disasters have impacted on the safety of people which have become important issues to consider by transportation professionals [1 - 3], inferring that planning a large-scale evacuation has become a necessity [4]. Examples where large-scale evacuations have occurred are the evacuation of the World Trade Centre, the evacuation of New Orleans when hurricane Katrina hit the US' south coastline in 2005 and the evacuation in response to the 2010's love parade stampede in Duisburg, Germany. Planning and running large-scale trial evacuations can be very costly [3, 5], both financially and in the number of people required to perform these test evacuations. These trials must be run a number of times to get a realistic picture of how the evacuation was performed: to get the optimal evacuation plan, different strategies and scenarios have to be tested. Moreover to physically evacuate a large area, such as part of a city, is not feasible and hence no proper evacuation plan can be obtained for these cases.

During live evacuations, incident commanders may have no idea which evacuation routes will lead the evacuees to safety in the shortest time. This is particular the case if these evacuations may not be necessarily from a built environment, where an evacuation plan may have already been devised, but from any open area which does not require an evacuation plan. In a built environment, people tend to evacuate from the exit they entered the domain [6] and if that particular route becomes inaccessible due to some damage to it, the evacuees may not be aware of alternate exit points. Incident commanders must be ready to cater for all the different scenarios that can arise during these evacuations. A solution to remediate all of these problems is the use of evacuation modelling. The evacuation model can be linked to CCTV and it is used to dynamically modify the procedures.

## *1.1 Evacuation modelling*

For decades, evacuation modelling developers have been designing and improving models that can predict evacuation movement and behaviour [6 - 15]. The use of evacuation / pedestrian modelling is well established as part of building design to ensure that buildings meet performance-based safety and comfort criteria [12, 16 - 22]. During live incidents, incident commanders can use evacuation models to test different evacuation scenarios to obtain the optimal evacuation plan [23]. To obtain the best evacuation plan, each simulation must be performed significantly faster than real-time so that the incident commanders have time to run a simulation, analyse the results, modify the scenarios, run the simulation again until the best solution is obtained. However, evacuation models such as the buildingEXODUS that discretise space and incorporate individual people and their interactions [12, 24] are computationally expensive [25]. Computational overheads involved in such simulations may mean that the time taken by the processor to run simulations involving very large populations or very large geometries (or both) may require several hours.

The evacuation model's predicted time to physically evacuate a geometry is called the *simulation time* and the computational speed of the processor to simulate the evacuation of that geometry is called the *wall-clock time* [26]. The wall-clock time should be considerably faster than the simulation time to be of any use to incident commanders. Evacuation models which do not use a formula to predict the movements of its population, will give out different simulation times at every run. This is due to the unpredictibility of the population's movement that the model promote. On the other hand, in deterministic models; such as in the field of Computational Fluid Dynamics, the same simulation times are obtained at every run.

The simulations of the evacuation of small geometries are usually done in a very short time. For example, using the buildingEXODUS evacuation software, the evacuation of 1,000 people from a supermarket, with the surface area of 4,000m$^2$ required a simulation time of 86.73s and the wall-clock time to simulate this case was only 3.12s. Here the wall- clock time is over 27 times faster than the simulation time. The evacuation of small geometries will not have the need of incident commanders planning a live evacuation plan, as usually these evacuations are fairly simple and straightforward.

On the other hand, large-scale evacuation may need live evacuations plans as incidents are unfolding. A good example is the evacuation of the world trade centre (WTC). The buildingEXODUS software was used to simulate the evacuation of the north tower of the WTC [27 - 29]. This involved a building of 110 storeys and two different target populations, one involving approximately 8,200 occupants and the other involving 21,000 occupants. With a population of 8,200, the actual evacuation required around 100 minutes in simulation time while with a population of 21,000 the evacuation may required approximately 150 minutes. Using a 3.6 GHz Pentium 4 PC with 3.25 GB RAM, these simulations required approximately 25 minutes and 120 minutes of wall-clock time respectively.

Even at four times faster than real time, the 25 minute simulation of the WTC evacuation is not sufficiently fast to be of use to incident commanders. Performance would need to be significantly faster than real time before it may prove useful in such applications, hence the need for a parallel version.

### 1.1.1 Evacuation software for large-scale evacuation

There are several evacuation models that can simulate large-scale evacuations and several that are restricted to simulate only small-scale evacuations. Some examples of models that are capable of simulating large-scale evacuations are the buildingEXODUS model (30,000 people in a 60,025 m2 area [30]), the PedGo software (80,000 people from a stadium [31]), Simulex [32], Legion [34, 35] and STEPS can simulate evacuations from stadiums [12, 33]. Some of the models which can only handle small-scale evacuations are the GridFlow model (can support up to 5,000 occupants [12]) and the EXITT model (can only simulate evacuations from residential buildings [12]).

There are different types of evacuation models, some which can predict just the evacuation times and others which incorporate human behaviour and external stimuli to study the various external influences (fire, smoke etc) in much detail [17] as well as predict the evacuation times. The ability to simulate the evacuation of large domains depends on how the models represent their regions and people. The more attributes there are to the regions and people, the more computationally expensive it is to load and run the simulations.

There are three main methods to represent a geometry by the models: the coarse node method, the fine node method, and the continuous method [36]. Models that incorporate the coarse node approach are not that demanding on the processors and can hence load and simulate large-scale evacuations [1]. A drawback for those models is that they cannot consider human behaviour and interactions in detail. These models tend to treat the movement of people like a fluid flow. Models that use a fine node and continuous approach are much more computationally expensive to load and run. Computational overheads involved in such simulations may mean that clock times for simulations involving very large populations or very large geometries (or both) may require several hours. Indeed simulations, involving extremely large geometries, such as those for urban-scale applications, if at all possible, may require days [16].

The buildingEXODUS software is one of the evacuation models that can perform large-scale evacuation, even though it is a fine node model. There is a limit to what extent buildingEXODUS can perform large-scale simulation as it will depend on the size of the geometry / population incorporated and the computer hardware used. The memory and processing capabilities of the PC used determine how large a simulation can be; more memory and processing power will imply a larger simulation can be formulated. A good alternative to buying a single high performance computer and simulating the large cases is to make use parallel computing. Parallel computing can either use multiple processors in a shared-memory environment or use several computers on a network, called distributed computing, to perform these large simulations. Parallel computing uses all the computers available as one very powerful virtual computer. Parallel computing is widely used in many different fields [37, 38]. Some examples are in the Computational Fluid Dynamics (CFD) field [39 - 41], fire modelling [42, 43], weather modelling [37, 44, 45], biology, search for extra-terrestrial life [46] and many more areas. However, very little literature is available on its use to evacuation modelling.

Please see Chapter 2 for a more detailed explanation of the various models and their attributes.

## *1.2 Requirements imposed by the software developers*

In creating a parallel implementation of the buildingEXODUS software, there are a number of core requirements that the parallel implementation must satisfy. These requirements are intended to make the parallel version of the software both flexible and easy to adopt by the existing users. These include the following considerations:

1. There should be no difference between the input or output files for the serial and parallel implementations of the software. This will allow applications to be designed and the results visualised using the familiar serial components of the software.

2. The parallel implementation should work on an arbitrary number of processors.

3. Minimal additional investment (in time and hardware/software) should be required by the user to effectively run the software.

4. The parallel implementation is intended to function in a Microsoft Windows environment.

5. Ideally the parallel architecture could be office PCs connected by a LAN network.

6. There should only be one EXODUS source code i.e. separate parallel and serial source codes would not be developed.

Having a single software source code is desirable in order to minimise the effort required to maintain the product [16].

## *1.3 Research Questions*

In starting this work, there were two main areas of research that needed to be explored and these are:

1. **Can parallel computing techniques be usefully applied to buildingEXODUS to enable faster wall-clock times and also to load vast cases, which cannot be loaded on a single processor?**

    - How much speed-up (How much faster a parallel version is to its serial version) can be obtained from using the parallel techniques?

    - By using all the combined memory of the different computers on the network, how big a scenario can be loaded and run?

    - Is the parallel system scalable? Will increasing the number of processors on the network increase the speedup?

2. **Which partitioning techniques will yield the optimal partitioning strategies?**

- How to partition the problem, so that the communication costs is minimised and the load balancing is preserved?

- Is a partitioning software better suited for partitioning the problem, or must a specialised partitioning algorithm be created?

- Is dynamic repartitioning an option to consider? Will extra communication and algorithmic costs be a limiting factors on the advantages gained from a dynamic repartition?

## 1.4 Objectives

The objectives of this thesis are to try to fulfil all the requirements imposed by the software developers and to resolve those questions asked in the previous section. This thesis presents a parallel approach to evacuation modelling in order to aid real-time, large-scale procedure development.

### 1.4.1 Can parallel computing techniques be usefully applied to buildingEXODUS to enable faster wall-clock times and also load vast cases, which cannot be loaded on a single processor?

The parallel version of the buildingEXODUS codes should be compared against the serial version to confirm that the parallelisation techniques have not modified the evacuation outcome which the serial version would have given. The parallelised version should be compatible to run on standard office PCs connected via a LAN network. The performance of the new version must be tested to see if there have been sufficient improvements in the wall-clock times compared to the serial version. Large problems, which were previously impossible to load and run on a single PC, should be tested on the parallel version to confirm its ability to load vast geometries containing a large number of people.

### 1.4.2 Which partitioning techniques will yield the optimal partitioning strategies?

A significant exploration of all the domain decomposition strategies must be researched in order to come up with the most efficient technique.

Whichever strategy is chosen, care must be taken to minimise communication costs amongst the processors, which otherwise will increase the wall-clock time. Another important factor that is crucial in maintaining a good parallel performance is load balancing. Much research must be done in optimising these two functions in order to obtain the optimal parallel decomposition strategy. A decomposition technique must be devised to both promote the load balance of the problem and reduce the communication costs associated with a parallel system. In the event of a

failure in finding a partitioning technique which can be applied to every evacuation scenario, a dynamic repartitioning approach must be researched. The advantages gained from using a dynamic approach must be compared to the resulting costs incurred in order to determine which are the crucial factors affecting this approach.

## 1.5 *Structure of Thesis*

**Chapter 1. Introduction:**

This chapter has introduced the research questions and goals of this thesis and the objectives to answer them by researching the subject.

**Chapter 2. Background and Literature Review:**

This chapter provides a state of the art description of the issues raised in chapter 1. Various topics required for the advancement of this research are elaborated. These topics are: evacuation modelling, parallel processing and partitioning strategies.

**Chapter 3. buildingEXODUS evacuation model:**

An overview of buildingEXODUS is described in detail in this chapter. The limitations of buildingEXODUS are introduced and how parallel processing can be a solution to address these shortfalls.

**Chapter 4. Parallel implementation of buildingEXODUS:**

The prototype used to work out the parallelisation strategies to be used on the buildingEXODUS is described. The software parallelisation strategies applied to buildingEXODUS as well as the rejected techniques are explained.

**Chapter 5. Domain partitioning in buildingEXODUS:**

The different decomposition strategies are explained and discussed. An intensive investigation was researched into which partitioning strategy is capable to partition any type of geometry and minimise the communication costs whilst maintaining load balance.

**Chapter 6. Dynamic repartitioning in buildingEXODUS:**

A dynamic repartitioning strategy is devised and tested in comparison to the results obtained when the static strategy was employed. Several avenues were explored to culminate in the final partitioning strategy adopted.

**Chapter 7. Conclusions:**

A summary of the current research in this thesis is outlined and the concluding recommendation on the outcome of this research is proposed.

**Chapter 8. Further Work:**

The suggested future work regarding the work present in this thesis is explained and subsequent research areas which can improve the findings are advised. Additional research questions were also raised in this chapter.

# 2 BACKGROUND AND LITERATURE REVIEW

## *2.1 Evacuation modelling*

Evacuation modelling is frequently used in a range of sectors [11, 17, 47, 48], such as in the naval, building, aviation and the rail sector [47]. Given the recent occurrence of several large-scale man-made and natural disasters [1, 3], the need for analysing a wider range of hazards (including terrorist attacks) [49] and as a result evacuation planning has become all the more important [50, 51].

The huge technical progress in computational power has made the application of evacuation models to large-scale incidents more viable [17, 52, 53]. Ekman et al. [54] concluded that computer performance growth was over 50% between 1985 and 1996 and has decreased to 41% in the 1996 - 2004 period. Meanwhile clock frequency has been improving at a rate of 29% annually with it being 16MHz in 1985 to peaking to 3.4GHz in 2004 [54]. Kareem et al. [55] states that the performance of modern processors is rapidly increasing with the number of transistors per chip, increasing exponentially over the last three decades; please refer to Figure 2.1. This conforms to Moore's law. Hence, with the current processing power available, many evacuation models can simulate the evacuation from small-scale environments such as low-rise buildings (under 22.9 metres [12, 13]), naval vessels and aircrafts [17, 47, 48]. However, due to high computational costs, not all models cater for the simulation of thousands of people in large-scale evacuations, such as from high-rise buildings, stadia, and even cities.

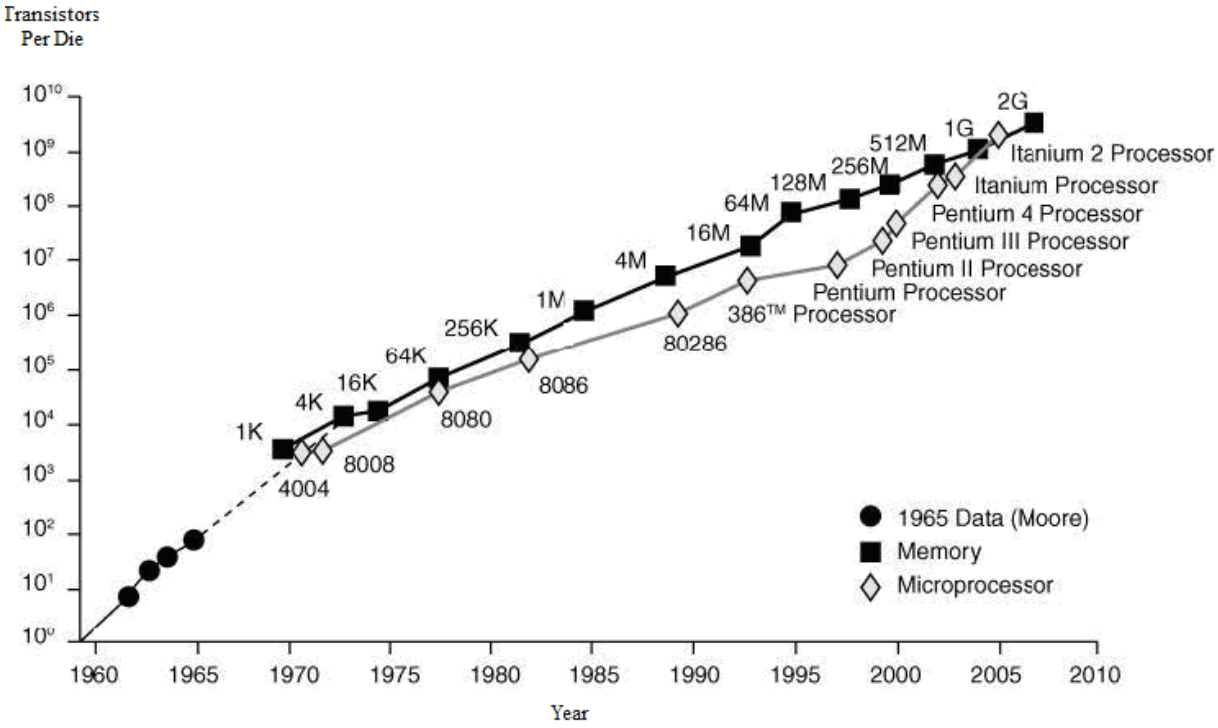



**Figure 2.1 - Scaling of transistors, the number of transistors is expected to continue to double about every two years, in accordance with Moore's Law [55].**

### *2.1.1 Benefits of evacuation modelling*

There are many benefits of using evacuation models and these are now discussed. Performing a full-scale evacuation poses considerable ethical, practical, financial [6, 25] and safety issues [53]. The use of evacuation models can address all of the above problems. Buildings, ships and airplanes commonly use evacuation modelling for performance-based analysis during their planning stage [10]. In this day and age, bigger and more complex structures are being constructed [6]. Bigger cruise ships (Voyager of the seas [56]), airplanes (Airbus' A380 [57]) and buildings are being designed [53]. The constructors can use evacuation modelling to gain valuable insights about the possible evacuation issues in the planning stages [53], when they still have time to work towards the best design to optimise the evacuation scenario. The gathering of large populations in one location can occur for a number of reasons: major sporting events which can hold up to 70,000 fans [58], large transportation terminals, such as New York Grand Central Terminal and Pennsylvania Station accommodate 200,000 passengers each weekday [58], religious pilgrimage, whereby over 2 million pilgrims go to the Hajj pilgrimage every year [59],

festivals and carnivals with more than one million spectators viewing the annual Tournament of Roses parade along a 9 km route in Pasadena, California [58], the Love parade 2002 in Berlin had about 500,000 people [60] are some examples. These require detailed planning to cope with the range of issues presented by the gathering of large populations. Organisers of these events can use evacuation modelling to plan the evacuation from these events.

### *2.1.2 Types of evacuation models*

There are various types of evacuation models employing different techniques and ways to represent their data. An evacuation model can be assembled from a choice of different modelling methods available and diverse ways to represent its geometry and its population.

A brief description of the different components of an evacuation model is given to better understand the core features of buildingEXODUS. Each of its features is important in choosing the appropriate parallel techniques needed to parallelise and partition the software.

#### 2.1.2.1 *Modelling Method*

According to Kuligowski et al. [12], evacuation modelling uses three main types of modelling methods: the behaviour model, the movement model and the partial behaviour model.

1. Behaviour model:

The behaviour models try to incorporate human behaviour in the process of evacuation. These models takes into account, not only the physical environment, but treats its population as active individuals which can respond to stimuli such as various fire hazards and have individual behaviour, such as reaction times, exit preferences etc. The buildingEXODUS evacuation model is an example of this type of model [6, 60, 61].

2. Movement model:

This type of model concentrates solely on the population capacity of the structure and its various components [6]. During the simulation, people are assumed to evacuate the structure immediately, ceasing any other activities. They all evacuate in the direction and speed of egress predetermined by physical considerations only [6]. These types of models ignore the population's individuality [6] and use an equation method [62] to treat their movement as a fluid [8, 12, 17] or gas-kinetic flow [17].

3. Partial Behaviour model:

The partial behaviour models primarily calculate the occupants' movement, but can have some behavioural characteristics incorporated [12]. Some behaviour can be represented by pre-movement time distributions, overtaking, individual characteristics and the effects of smoke on the occupants [12]. An example of such a model is STEPS (Simulation of Transient Evacuation and Pedestrian MovementS) which is a movement / partial behaviour model and developed by the Simulation Group of Mott MacDonald [12, 63 - 66].

2.1.2.2 *Enclosure representation*

There are three main ways that an evacuation model can represent its geometry: the coarse node network, the fine node network and the continuous network. It is also possible to have a hybrid method that mixes these three approaches, currently EXODUS is developing a hybrid version which can merge the three types of enclosure representations to build a geometry [67].

1. Coarse node representation:

The whole region is divided into partitions, which can represent rooms, corridors, stairs etc. Each partition becomes a node and are linked to its neighbouring node by an arc. Each node has various attributes associated with it, such as the number of people it can hold, the flow rate of its population and calculates the motion of the people inside it by some formula. In these models, occupants move from segment to segment, and their precise position is less defined [68]. In models which incorporate the coarse node network, most social interaction elements are made

inconsequential because evacuation times depend primarily upon node capacity and the traversal times [14]. Hence, the positions of the people, detailed spatial representation and also the people-people interactions are not specifically represented. However, these models can simulate large-scale evacuations as they are less computationally expensive to execute [17]. Figure 2.2 illustrates this concept.

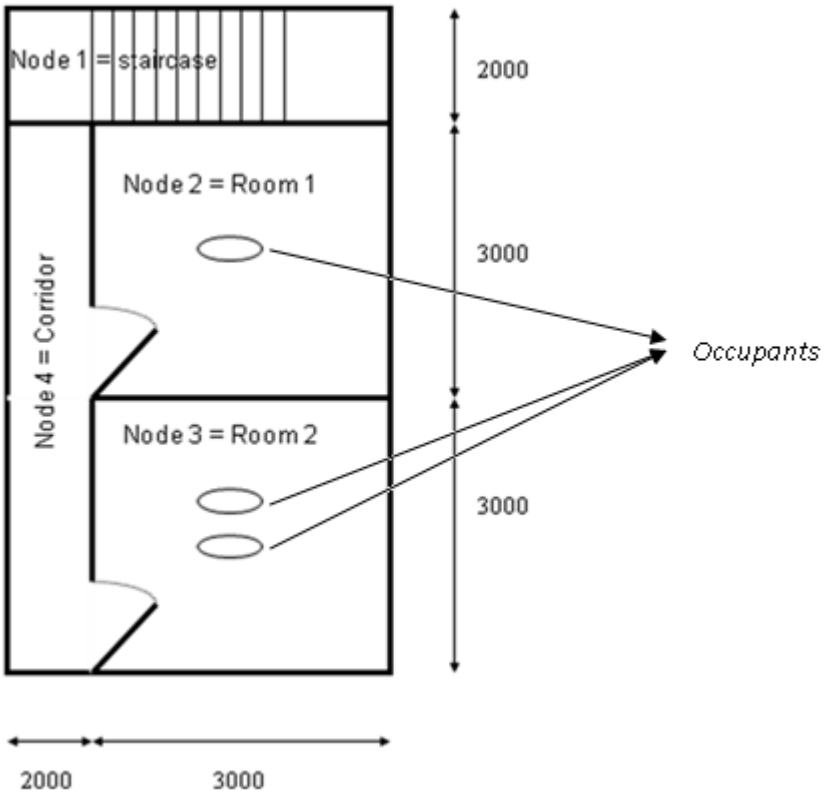



**Figure 2.2 – Adapted from the coarse node representation from [48].**

2. Fine node representation:

In a fine network representation, the whole area is discretised into a collection of nodes / cells. The size of each node varies from model to model. For example EXODUS uses a 0.5m x 0.5m node spacing, STEPS uses a 0.5m x 0.5.m cell [12] and the CAFE model employs 0.4m x 0.4m cells [76] whilst EGRESS employs hexagonal cells [12, 69]. Usually each node can hold one person and each node is connected to its neighbouring nodes by arcs. Hence, each compartment within a geometry is made up of several nodes. This fine representation also enables an accurate

representation of an enclosure as well as representing obstacles in that enclosure [48]. In these models, individuals can be easily located and tracked during the simulation. However, not all models that incorporate the fine node network can perform large-scale evacuation since it is computationally expensive to represent each node which can contain several attributes. Figure 2.3 demontrates the fine node network to the same enclosure used in Figure 2.2 to illustrate the coarse node network.

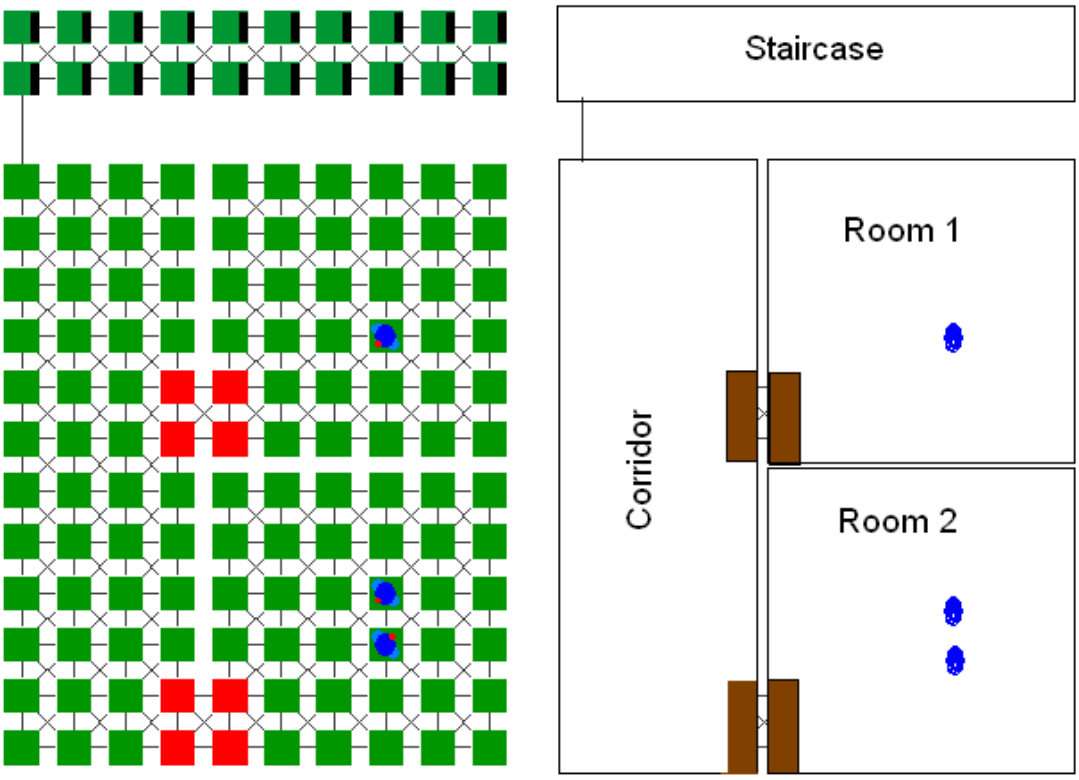



**Figure 2.3 – Fine Node representation of the enclosure in Figure 1.**

3. Continuous network representation:

A continuous model applies a coordinate representation of the area, where each point in that domain has a co-ordinate; such as Simulex [70 - 72], Social forces model [94], Legion [12], and Pathfinder [73]. The occupants can move from one point in space to another throughout the

enclosure [12]. These models can find it very computationally expensive to simulate large-scale evacuations [84].

#### 2.1.2.3 *Population representation*

There are two ways that an evacuation can represent its population: the Global or the individual approach [12].

i. Global representation:

These models represent its population as a homogeneous population. Their motion can be compared to the flow of fluids, the flow of granular materials, gas-kinetic models [17], and even to the progression of hazardous substance or smoke [14]. This type of global representation cannot distinguish individuals during the simulation. The movement of the population is determined by densities, usually a mass density derived from the people's positions and a corresponding locally averaged velocity [17]. Models that features a global representation of its population are not computationally expensive to run and can hence simulate very large population [74].

ii. Individual representation:

Models encompassing individual representation tracks the movement of each occupant throughout the simulation. At any point in the simulation, the model can provide information, such as the current position or any personal attributes the user might want about each occupant [12]. Usually these models have many personal attributes, such as age, walking speed etc, associated to each person. For example, "EXODUS simulation program furnishes perhaps the most complete set of social psychological attributes and characteristics for each agent, twenty-two in all." [14]. Due to the number of attributes associated to each occupant, the models can only represent as many people as the computational memory available to the user will allow. Hence, most models employing the individual representation is restricted to the number of people it can simulate.

## 2.2 Parallel computing

Currently, many applications utilise more computing power in terms of memory and performance than a sequential computer can offer. An example of such an application is in the field of weather modelling where there are vast data to be analysed and modelled for large regions of the earth. Parallel processing can provide a solution to this problem by linking a number of CPUs together connected via a communication system to enable data transfers. Parallel processing is being used in a variety of fields, such as in the CFD field, weather modelling [37, 44, 45], searching for extra-terrestrial intelligence [46] and many more areas. There are two main types of parallel archictures: multi-processor computers having multiple processing elements within a single machine, while clusters, massively parallel processors (MPPs), and grids use multiple computers to work on the same task. Since the requirement imposed by the software developers is to use office PCs connected by a LAN network, the cluster architecture was chosen for this research.

Recently multi-core processors, which are the inclusion of two or more complete computational cores within a single processor [77], were developed and are easily available [78]. This technology was conceived when engineers realised that Moore's law of doubling the performance of micro chips every couple of years is no longer physically feasible [77, 79]. Burger [77] states that compared to a single core processor, a multi-core processor with two cores can increase the processing speed by 93%. This technology is appealing as a means to improve the speed performance of buildingEXODUS. However, using a multi-core processor alone does not resolve the problems posed by loading and running large-scale simulations as the memory available is still similar to that of a single core processor. However, using a cluster of multi-core processors on a distributed network is an attractive option to consider. Multi-core and other shared memory devices within a single computer share the same memory and memory bus. This has the advantage of simplifying the parallelisation process but will suffer from memory bus contention that may restrict the potential benefits offered by the extra processing power available. It should also be recognised that a software designed to run in a distributed fashion will also work on multi-core processor.

## 2.3 *Terminology*

### *2.3.1 Speedup [37]*

Speedup is a measure to evaluate the performance of a parallel system over a sequential implementation. It determines the relative benefit of solving a problem in parallel. Speedup, denoted by $S_p$, is defined as the ratio of the time taken to solve a problem on a single processor to the time taken by a parallel system of p processors.

Speedup is defined by the following formula:

$$S_p = \frac{T_1}{T_p} \qquad \textbf{Eq (1)}$$

Where:

*p* is the number of processors

$T_1$ is the execution time of the sequential algorithm

$T_p$ is the execution time of the parallel algorithm with p processors

Theoretically, Speedup cannot exceed the number of processors p used. In most cases $S < p$ due to the costs involved in the parallelisation process. However, if a speedup greater than p is observed, this is called *superlinear speedup*. This phenomenon sometimes happens when there are hardware features that put the serial implementation at a disadvantage. An example of such a case is the size of a problem might be too large to fit in the cache of a single processor, hence degrading the sequential performance. The same problem partitioned into smaller parts, will be small enough to fit into each processor's individual cache.

2.3.1.1 *Scaled Speedup*

The speedup obtained when a problem size increases linearly with the number of processors is termed as a scaled-speedup. A parallel system is considered scalable if the scaled-speedup curve is close to a linear curve representing the number of processors [37].

### *2.3.2 Efficiency [37]*

Ideally a speedup value equal to p processors is perfect. However, in practice this is seldom achieved while executing a parallel algorithm. The processors involved cannot dedicate 100% of their time to the computations as there are communication and synchronisation costs involved and also times where processors will become idle.

Efficiency (E) is a measure to determine how a processor is usefully employed. It is defined as the ratio of speedup (S) to the number of processors (p) used.

$$E = \frac{S}{p} \qquad \textbf{Eq (2)}$$

In an ideal parallel system, efficiency is equal to one but in practice, $0 \leq E \leq 1$ depending on how efficiently the processors are utilised.

### *2.3.3 Amdahl law [80]*

According to Amdahl's law, if there is a fraction of the code *(f)* which is parallelisable with no scheduling overhead, then the speedup is governed by the following equation:

$$Amdahl's\ expected\ speedup = \frac{1}{(1-f)+\frac{f}{n}} \qquad \textbf{Eq (3)}$$

*n*: Number of processors

*(1-f)*: Fraction of the code which is totally sequential

### *2.3.4 Parallel SimTime[1] Performance Ratio*

While the computational speedup is important in determining the performance of the parallel system, another consideration is simulation time. How long it takes to run a simulation is the consideration which will determine how practical it is to use as an advice tool by incident commanders for large building based scenarios: as part of an interactive emergency signage system, a training tool in an interactive desktop environment and as a planning tool for large scale urban applications. A parallel SimTime performance ratio is used in this case to demonstrate how fast a computer simulation can render results compared to the actual time to evacuate a geometry.

## *2.4 Parallel architecture*

There are many types of parallel platforms, each with its own characteristics. The two main types of parallel processing architectures are: parallel processors where different processors share the same memory of that computer, and distributed computing where each computer is distinct, with its own processing element together with its own memory. The following are the main types of parallel architecture currently in use:

- Shared-memory multiprocessors or parallel processors that have more than one processor connected together that share the same memory [96].
- Parallel vector processors are designed to handle arithmetic operations on vectors. Although this type of processor was often used to build supercomputers in the past, they have lost their popularity due to the high cost [96].
- Distributed memory massively parallel processors (MPPs) consisting of many individual nodes, each essentially an independent computer in itself. Each node consists of at least one processor, its own memory and is connected to a network which links all the nodes

[1] **SimTime:** Simulation time (See Section 2.3.4)

together. Traditional MPP systems may contain hundreds or thousands of individual nodes and they have custom networks which are very fast [97].

- A computer cluster consisting of several distributed memory computers built from mass produced PCs and workstations connected via a network [96 - 98]. Cluster computing is now a popular platform for cost-effective parallel computing [99 - 101]. They are much cheaper than traditional MPP systems [102], but are more difficult to use since the network capabilities are currently much lower. The number of nodes used is much lower, most often involving fewer than 100 computers [97].

- Multi-core processors contain two or more computational cores. Each core is complete with its own hardware but share the same memory. By dividing the work between two execution cores, a multi-core processor can perform more work within a given clock cycle. Multi-core technology allows servers to run tasks in parallel that previously would have required multiple processors. [77]

- Network / Internet / Grid computing uses processors from different geographical area, connected together via the internet. Networking computing is not very effective due to the low communication rate of the network, large latencies, and differing CPU rates of connected computers [96].

- GPU (graphics processing unit) computing is the use of a GPU to do general purpose scientific and engineering computations [103]. Both the CPU and the GPU are used as a co-processing computing model, where the computationally intensive side is accelerated by the GPU and the sequential part taken care by the CPU [103]. This innovation is rapidly out-performing the results obtained from using parallel computing [103 - 105]. The downside of this technology is that it may be hard to scale to very large cases due to the memory constraint on a CPU [104]. This technique, though an attractive option to consider, cannot be used in this current research due to the high level of communications and synchronisation required by an evacuation model. The GPUs cannot directly communicate to each other [106] and hence unable to cater for the necessity of communications and synchronisation needed to allow the movements of individuals at the geometric boundaries.

Due to requirement 5 (Section 1.2) imposed by the software developers that "the parallel architecture used for this project will be office PCs connected by a LAN network", the cluster category is the most appropriate architecture to adopt. Each PC on that network has its own local memory and will need to share information using a parallel communication package. This type of parallel architecture is highly scalable as increasing the number of PCs combines their individual memory load to handle larger sized problems with higher memory requirements. However, this type of architecture is harder to program than systems that use a shared memory aproach which also significantly reduces comunication costs [107] whereas intensive data communications amongst the PCs in a distributed memory approach can cause a bottleneck in the communication link. Although the development of the parallel implementation is initially intended for connected office based PCs it would potentially be simple to port this to other parallel architectures once developed.

## 2.5 *Parallel algorithm models*

There are various strategies that can be used to parallelise a program. The type of the problem significantly influences the strategy selected. The different models that can be used are explained below.

- Data Parallel model [37]: Data parallelism occurs when the tasks are mapped onto processes and each tasks performs similar operations on different sets of data. Data parallelism algorithms can be implemented on both shared-address space and message passing paradigms. A key characteristic of data-parallel problems is that for most problems, the degree of parallelism is proportional to the size of the problem. Hence the system becomes highly scalable; adding more processes enables the solving of larger problems. This strategy will benefit the type of problem that is being investigated in this thesis. For an evacuation model, a possible decomposition of the task can be the physical layout being partitioned into sub-regions which are assigned to a processor. Another possible decomposition of the task can be the division of the population into the number of processors available and assigns each sub-population to a processor. An example of this type of application is the matrix manipulation problems [108].

- Task graph model [37]: This type of model is more suitable for tasks that are closely related. The interelationships among the tasks are utilised to promote locality and reduce interaction costs. This model is more suitable for problems with large data and less computations and favours the use of shared-address space. This type of strategy will not benefit an evacuation simulation which will involve a large independent population and intensive complex calculations about their movement coupled with other factors affecting the occupants. Applications of this model is in the scheduling of real-time tasks [109] and in the image / video compression / decompression field [110].

- Work pool model [37]: This model performs a dynamic mapping of tasks onto processors for load balancing. This strategy is more suited for problems which have a small amount of data compared to the amount of computation. The reason is that each time the dynamic mapping occurs, data needs to be reallocated. A large amount of data will incur too much communication costs at each mapping. Consequently, this model is not appropriate for an evacuation model which can have a large population together with a large domain. An example of this model is a matrix-vector multiplication [111].

- Master–slave model [37]: This type of model involves a master process giving out jobs to slave processes and retrieving them. In this system, the slaves are continuously being kept busy as the master is constantly handing out new jobs when each slave becomes idle. However, this type of strategy is more suited for problems which have independent jobs that do not need to be shared with other slave processes. A pedestrian evacuation simulation requires too many interactions and this type of parallelism will not be suited to it. However, the idea of a master processor allocating the work at the start and retrieving them at the end can be an interesting avenue that can be adopted for an evacuation model. Quinne et al. utilised this paradigm in the parallelisation of the Social Forces model [89].

- Pipeline model [37]: This type of model involves each processor working on some part of the algorithm. For example, for a pedestrian evacuation algorithm, one CPU can handle the movement of the individuals, while another CPU can handle the hazards that these people are being subjected to. This scheme will not be suited for the pedestrian evacuation software as all the different sub-algorithms are about each person and they will need to be merged at each time step and thus the communication costs will be too

high. Another setback is that each CPU must hold the whole of the geometric domain, thus employing much memory requirement. An example where the pipeline model is being applied is in the model-based image coding algorithms [112].

- Hybrid Model: This model can merge any of the above models and use some of their capabilities. A good example that can apply for an evacuation software is merging the data parallel model with the master-slave approach. This newly hybrid model can have a master processor which initially partition the data (according to the data parallelism) and allocates the tasks to the slave processors. The master can work on the tasks itself and finally retrieves and compile the results after the simulation.

## 2.6 *Related Work*

Much research on the parallelisation of movement models have been studied in the field of traffic simulation [81 - 87]. However, not much literature can be obtained about the parallelisation of a pedestrian / evacuation software. As in pedestrian movement models, which can represent their occupants either as a homogeneous mass or as individuals, many traffic models have incorporated the microscopic approach of representing its agents as individual objects interacting with each other, rather than simulating the traffic as a fluid flow [83, 84]. Similarly to how evacuation modelling is used to ensure buildings meet performance based safety and comfort criteria, traffic modelling is routinely being used in emergency planning and management in order to decide on a proactive evacuation [88]. The only literature found concerning the parallelisation of a pedestrian evacuation software similar to what is being researched in this thesis were the works of Quinn et al. [89] and of Karavakis [90]. PEDFLOW, a multi-agent simulation system designed to represent conflicting pedestrian flow at a detailed level, has been implemented in a parallel processing language but uses a shared data structure approach [91]. As is MassMotion, a suite of software tools that analyses and represents behaviour of individuals in the built environment, which has a 64-bit Multi-threaded design taking full advantage of multi-core processors and hence a shared-memory approach [92, 93], unlike the distributed approach being researched for this thesis.

2.6.1.1 *Review of "The parallel implementation of the social forces Model"*

A review of Quinn. et. al's paper titled "The parallel implementation of the social forces Model" [89] is presented in this section. The social forces model [94] is a pedestrian evacuation simulator that represents the population microscopically. Quinn et al. compares the temporal performance of the Cellular automata (CA) and social forces models. In both models, every person is individually represented and they have distinct attributes associated. The Social Forces model uses the continuous network to map its geometry, unlike the CA models which discretise its region into cells (40 cm on a side). Hence, the disadvantage of the social forces model is that it takes much longer to update the position of each individual compared to the time the CA model takes to update the position of the same population. Experiments have shown that when both models were used to simulate the evacuation of the same population, the social forces model was about 100 times slower than the CA model. Quinn et al. resolved this problem by parallelising the social forces model.

Their design process was to use the master / slave approach. The master reads in the geometric layout, scenario and population details and send these information to the appropriate slave processor. At each time step, the master process gathers data from the slave processes and passes this information to the appropriate rendering engine. The slave processes are interconneted on a 2D virtual grid. Each worker has part(s) of the domain allocated to it and at each boundary of the domain, "Ghost cells" are created and they represent the boundaries of the neighbouring processes. When occupants move within the domain controlled by one process, no communication is necessary to other worker processes. However, if an occupant has reached the boundary of the domain, then necessary details about that occupant needs to be communicated to the appropriate neighbour process and when the occupant has moved out of the main domain controlled by a process, then that process must hand over that person to the neighbouring process. The Message passing interface (MPI), as explained in Section 2.6.1.5, was used to enable the interprocessor communications.

The developers suggests that the shape of each sub-domain is encouraged to be as square as possible as this decreases the communication overhead and thus maximising the speed-up of the parallel program. The parallel implementation of the social forces model was benchmarked on SWARM, a linux-based multicomputer consisting of 2.4 GHz Inter Xeon CPUs conneted by a

gigabit Ethernetswitch. The test case simulated the evacuation of 10,000 people from an airport. Since the airport layout was long and narrow, the region was partitioned in a 1D grid. Encouraging speed-up was achieved and the system was scalable, with ten worker processes and a master process (11 processes overall), the system was able to update the positions of the 10,000 pedestrians nearly 50 times per second. Figure 2.4 shows the results obtained when different number of processes was used.

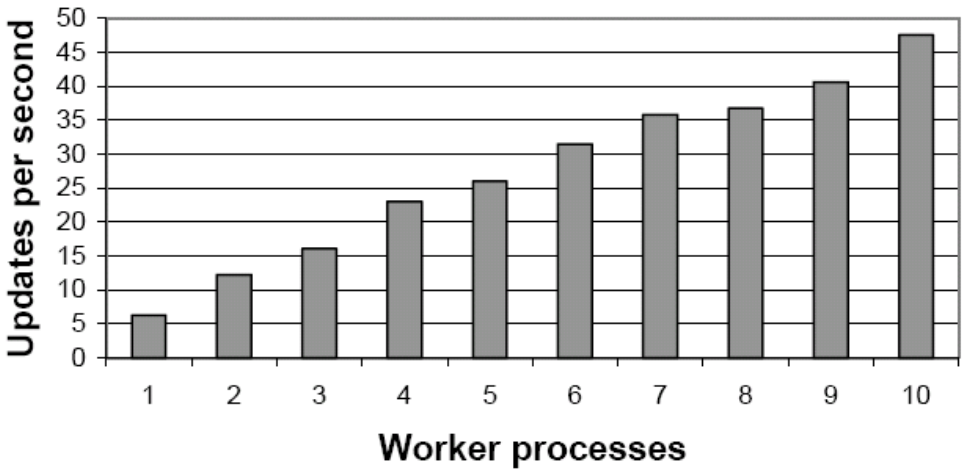



**Figure 2.4 – Frequency at which the positions of 10,000 pedestrians are updated, as a function of number of worker processes. Each worker executes on one CPU. One additional CPU is allocated to the manager process. [89]**

The parallel implementation achieves good results if the pedestrians are evenly distributed across the area as each worker process contains roughly the same sized domain. However, if the occupants are concentrated in one area, then some processes take longer to complete one time step, thus slowing down the whole system's parallel execution time which is determined by the processor finishing last. An unbalanced workload can reduce or even eliminate the performance benefit of adding more processes. The developers believe more research needs to be done to optimise the speed-up by finding the most appropriate partitioning technique to divide its domain.

While the updates of the positions of the population has improved considerably from using the serial version, no information about the speed-up (How much faster a parallel performance is compared to its corresponding serial version) is made available. A considerable improvement in

the position updates of the population does not necessarily imply a similar improvement in the parallel performance of the process and would tend to give an optimistic view of the potential speed-up. The parallel performance may be adversely affected by the communication costs the system has incurred or by the load balancing of the problem, thus counteracting the benefits of the improvement achieved in the position updates. The ultimate goal of the parallelisation of a problem is to achieve good speed-ups and no mention about that outcome was made.

#### 2.6.1.2 *Review of "A distributed analysis and monitoring framework for the compact Muon Solenoid Experiment and a pedestrian simulation".*

A review of Karavakis' dissertation titled "A distributed analysis and monitoring framework for the compact Muon Solenoid Experiment and a pedestrian simulation." [90] is carried out in this section. Legion Studio has been modified to work in a parallel processing framework [89]. The pedestrian software mentioned in this section is called Legion Studio and has a commercial worldwide usage. Similar to EXODUS, Legion Studio is both a behaviour and movement model which represent its population microscopically. However, it represents its geometry with a continuous space description which is computationally more expensive to run. Even with these expensive configurations, Legion Studio is able to simulate large areas such as airport terminals, sport venues, shopping centres and venues for major international events such as the Olympics. To help with running these large cases efficiently, Legion Studio adapted one of its application to employ the benefits of multi-threading capabilities of multicore PCs as well as the ability to perform on a distributed environment.

The Legion Studio pedestrial simulation software comprises of three applications: the Model Builder, the Simulator and the Analyser. The Model Builder creates a model for the space needed for the simulation; the Simulator is used to run a simulation of how pedestrians move within that space; and the Analyser is used to run a series of analysis on the simulated space. The Analyser can used both an On-line analysis of the simulation as well as an off-line analysis of a recorded simulation. The only difference between the on-line and the off-line Legion analysis is that during the on-line analysis, the Analyser communicates with the simulator. Unlike the current research for the development of a parallel version of the whole EXODUS software, only Legion Studio's

Analyser application was adapted to benefit from the use of Multi-threading and distributed computing.

#### 2.6.1.2.1 Multi-threaded Legion analyser

Similarly to MassMotion's [92, 93] use of multi-threaded capabilities, Legion's Multi-threaded Analyser creates a thread pool with a size equal to the number of the CPU cores. This implementation has managed to improve the performance of the software as well as optimising memory management. Six models with different levels of complexity and size were used on a 2 GHz of CPU dual-core system with 2 GB of memory to benchmark the multi-threaded application. The times published as well as a calculated speedup are illustrated in Table 2.1:

**Table 2.1 - Benchmark values for the Multi-threaded Legion Analyser**

| Models | Entities | Simulation time | Total Time HH:MM:SS | | Speedup |
|---|---|---|---|---|---|
| | | | Original | Multi-threaded | |
| PM Peak (Small-sized) | 350 | 3hrs | 00:39:45 | 00:17:43 | 2.24 |
| UP Demo (Small-sized) | 552 | 1hr | 00:07:50 | 00:05:08 | 1.53 |
| Gatwick Airport Station Re-development (Medium-sized) | 1200 | 1hr | 00:22:32 | 00:09:31 | 2.37 |
| New WTC Model (Medium-sized) | 2500 | 1hr30mins | 01:41:22 | 00:34:58 | 2.90 |
| London Olympic Park 2012 (Large-sized) | 51000 | 14 mins | 02:16:25 | 01:29:50 | 1.52 |
| HOS Case3 (Large-sized) | 52000 | 19 mins | 01:25:04 | 00:57:41 | 1.47 |

Using the multi-threading capabilities of both cores of a dual-core PC realise better speedup values compared with the results obtained if only one core was utilised. It can also be observed that the speedup is very problem dependent and these results represent low scale parallelism (2 processors). The attraction of using both cores with the added benefits of sharing the same

memory, thus reducing the communication costs, is very encouraging option to consider for this current research. However, using both cores from the dual-core processors may potentially incur memory bus contention and thus degrade the performance of the parallel system.

#### 2.6.1.2.2 Distributed Legion analyser

The main objective for developing a distributed version of the software was to allow the simulation of larger and more complex cases. In the early stages of development, the OpenMP standard [95] was considered, but was abandoned because OpenMP can only be used in a shared-memory environment. Similar to what is being designed for the parallelisation of buildingEXODUS, the distributed-memory approach was implemented by Legion Studio by using a master-slave strategy connected via a network. As stated by Karavakis that MPI being the most popular message passing library for parallel computing, the same library as well as a master-slave approach is being strongly considered for this current research.

The London 2012 Olympic park populated by 56,500 entities was used as a test case to benchmark their distributed version. However, the published values are not the total time it takes to analyse the whole simulation but a second of the total time involved to analyse the whole problem. Figure 2.5 shows how scaling the problem by adding more processors favourably improved the time it takes to analyse the above population in one second.

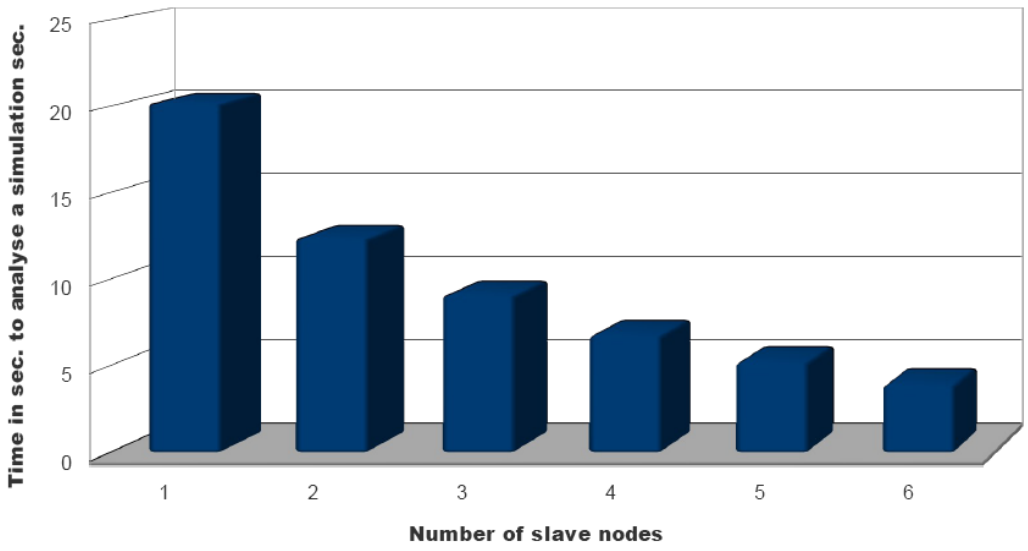



**Figure 2.5 - Time in seconds to analyse a simulation second. [90]**

From the graph above, a very encouraging trend is apparent in the performance of the distributed system when one second is being analysed. However, like Quinn et al. [89], these values do not offer a true representation of the performance of the whole simulation but only a time-step of it. During a whole simulation there are different levels of computations involved, for example it may be more computationally demanding to simulate congestion compared to just having free-flow movements. In addition, a load imbalance generated by an uneven population distribution during a simulation can seriously hinder a parallel system. A time step out of a whole simulation cannot represent these factors. The results above do not reflect the complete picture of the simulation and hence a true representation of the speedup of the whole system cannot be measured. It would have been interesting to learn about the difference in the actual original and distributed total time it took to run the case, as published for the Multi-threaded version.

2.6.1.3 *The parallelisation of traffic models*

Traffic simulation models can use both a macroscopic model which characterise the traffic flow with fluid dynamics approaches [83, 84] or a microscopic approach which deals with the individual behaviour of each agent [81, 83, 85]. Due to the high fidelity of the microscopic model, computational demands of large-scale traffic simulation can be high, leading to the need to parallelise the models [83, 84]. The parallel architecture used can vary from using supercomputers [83] to a modest cluster of 10 - 20 standard PCs connected via standard LAN technology [81]. Several methods can be used for the parallelisation strategy, where the domain decomposition is a common choice. A domain decomposition can be performed to the area and the agents moving in or out of the sub-domains need to be communicated across [81, 84]. Another possible strategy is to use functional decomposition where different modules can run on different computers [81]. For example, the micro-simulation can run on one computer while an on-line routing module could run on another. It was found that while the functional decomposition technique was easier to implement, it poses severe limitation on the speed-up achievable [81]. Due to the easier accessibility to the literature about TRANSIMS (Transportation Analysis and Simulation System) [81, 82], a more detailed review is presented in the next Section 2.6.1.3.1.

2.6.1.3.1 Review of the parallelisation of TRANSIMS

TRANSIMS [82] was originally developed by Los Alamos National Laboratory to run exclusively on a Linux cluster environment. It performs a domain decomposition (into sub-domains) by using METIS (a partitioning software) and uses MPI (a communicating package) to communicate the movement of the agents passing on the boundaries of those sub-domains. TRANSIMS Version 4 (V4) was designed and it allows the software to run efficiently on powerful Windows desktop machines and added many new capabilities. However, TRANSIMS V4 is unable to use the processing power of multicore and cluster systems, which are necessary to run highly detailed models of entire metropolitan areas, in a reasonable amount of time. TRANSIMS are researching into moving from using desktop machines towards running it as a remote, but very fast cluster application. Following the parallelization of the microsimulator and

router, mechanisms for effective load balancing need to be found and implemented to achieve additional performance gains. The parallelization research is expected to continue for some time before the TRANSIMS software can make full use of the available cluster hardware.

#### 2.6.1.4 *Summary of findings*

For this current research, there are two main decomposition methods that are attractive options: a geometric decomposition and a load (population) decomposition. The likeliest Parallel algorithm model to apply for this research is the hybrid model, making use of the data parallel model together with the master-slave approach. In the geometric decomposition, the problem itself is divided amongst the processes and each process solves its part of the problem by using the same algorithm. For example in the case of the simulation of a pedestrian evacuation software, the whole geometry will be partitioned into sub regions and each CPU will simulate the movement of the people in its sub-domain. Communications will only happen when there are people crossing over the boundaries. An additional advantage of this paradigm is that the domain size will be significantly reduced, thus lightening the memory requirement of each CPU. A load decomposition involves the problem to partition its computational load, rather than its spatial domain. This method can be computationally expensive to adopt due to the constant interactions of the individuals and also may pose a limiting factor to the size of the problem since each processor must hold the whole geometric domain. Both methods are further elaborated in Section 4.2.3.

#### 2.6.1.5 *Communication Package*

There are a number of free communicating packages that support Windows, such as PVM (Parallel Virtual Machine) and MPI (Message Passing Interface). Both the PVM and MPI packages have their own advantages and disadvantages, but MPI is the standard, which is widely implemented [37] and is used nearly everywhere [113, 114]. It is generally considered that PVM supports a heterogeneous environment and not MPI. However, MPI specification is designed to encourage heterogeneous implementation, which both the MPICH and Local Area Multi-

computer (LAM) implementations support [113]. The MPI package provides a richer source of communication methods than PVM [115].

## 2.7 Domain Decomposition

Domain decomposition can involve the decomposition of partial equations in the solving of equations or can be used to partition a domain or graph [116]. There are domain decomposition algorithms to partition the former and other partitioning techniques for the latter problems. Due to the nature of this dissertation whereby there is a physical domain to be partitioned, the term 'Domain decomposition' referred from this point forward is intended for this sort of problem. Domain decomposition is the partitioning of the physical domain into a set of sub-regions where each sub-region is allocated to a processor. Each processor solves the problem within its sub-region and communicates with other processors only to update the elements located on the boundaries [89, 117, 118]. Many applications use this approach of decomposition, namely in the field of traffic modelling [81], pedestrian evacuation modelling [89, 90], and particle tracking [117, 119]. An optimised domain decomposition technique can play a crucial part in the performance of the parallel implementation. The perfect domain decomposition strategy is when the load balance is maximised and the communication overhead is minimised. The parallelisation strategy must be formulated in such a way that the amount of data that needs to exchanged is kept as small as possible while ensuring that all processors are busy at all times. However, this compromise is rarely achieved, and there are always a trade-off between the load balancing and the communication costs [117, 118, 120, 121].

### 2.7.1 Domain Decomposition techniques

There are various methods that can be used to partition a physical domain. There are dedicated partitioning software such as METIS [122, 123] and JOSTLE [124] which use similar methods and algorithms to partition graphs. However, not all parallelised problems use these dedicated partitioning software and they have devised their own partitioning strategy to suit their problems. Common techniques used for domain decomposition are detailed below:

- The Recursive bisection technique [81, 117, 118] initially splits the domain into two sub-regions with the same load. This procedure is recursively applied until the number of sub-regions is equal to the number of processors available.

- Standard Graph partitioning approach is a technique that partitions the vertices of a graph based on some criteria which is most beneficial for the problem type. For example if the domain is made up of nodes interconnected by arcs / links, a possible criteria is to divide the number of nodes into the number of available processors and assign each sub-group of nodes to each processor [120].

#### 2.7.1.1 *Approaches used to improve load balance and minimise communication costs*

Different approaches were applied by Quinn et al. [89] and the Transportation Research and Analysis Computing Center [82] to try to preserve either the load balance or minimise the communication costs associated with parallelisation. However, this compromise is hard to attain and ongoing research is still being done to optimise either of these two factors or both [82, 89].

##### 2.7.1.1.1 Techniques used to optimise load balance

Recursive bisection or even the standard graph partitioning can obtain a good load balance if only a single quantity (physical domain only) needs balancing. However, many types of problems include a physical domain which in turn contains particles / elements residing on them [81, 89, 117, 125, 126], such as the type of problem that is being researched for this thesis. These problems require that multiple quantities be load balanced simultaneously. For parallel efficiency, all processors must hold roughly an equal number of grid cells / nodes and approximately the same the number of particles / elements [117, 127]. These partitioning methods provide a good load balance if only a single factor, such as the physical region needs balancing. However, they do not consider multiple factors that make up the domain.

A particle decomposition approach was investigated by Hatsky [119]. The whole spatial grid is assigned to each processor which simulates only a subset of the population. This strategy was found to promote the load balance and reduce communication costs. However, higher memory

consumption was needed per processor. This technique can only be advantageous for small sized problems and where no interactions are required between the particles.

#### 2.7.1.1.2 Techniques used to minimise communication overhead

It was found that while partitioning a region, by making the sub-regions as square as possible [89, 117, 118] reduces the communication costs. Another technique is applying a bounded neighbour method [118], which is a domain decomposition technique whose main goal is to reduce the overhead introduced by partitioning the domain. This method also allows the user to bound the number of neighbours of each sub-domain, consequently this also bounds the number of communications of each process and thus reducing the overall communication overhead.

#### 2.7.1.1.3 Possible compromise between load balance and communication overhead

By applying weights to nodes / cells and to links / arcs, a possible optimised decomposition technique can be achieved [120]. Some partitioning software, such as METIS, use this technique [124, 128, 129]. The weights associated to the nodes / cells can be used to represent the load and the weights associated to the links / arcs can correspond to the communication costs. The following principle used by ParMETIS [123] shows good potential for partitioning an evacuation model.

Consider this region containing particles in Figure 2.6:

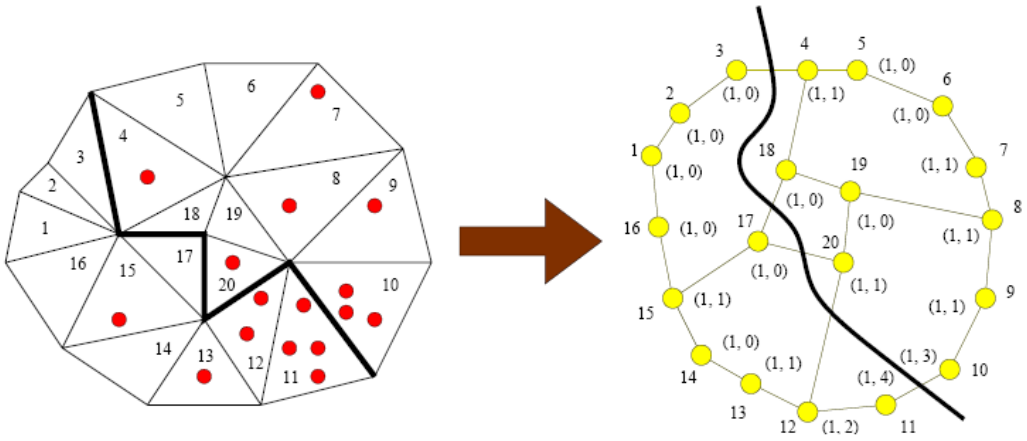



**Figure 2.6 – Graph containing sub-domains containing elements and how weights can be associated to the vertices to represent the elements [123]**

Each vertex has two weights associated to it; the first weight represents the work associated with the mesh-based computation incurred for each element (assume they are all ones) and the second weight represents the work associated for the element-based computation (the number of elements that are in that sub-domain). For example at sub-region 11 in the first graph, there are four elements in it and the weights associated for vertex 11 is (1, 4). A sum of the different weight categories can be calculated and then divided among the number of processors so that each processor gets roughly the same load in terms of weighted nodes.

A similar association that can be made to an evacuation model is that the vertices in the above graph can represent nodes and the elements representing the population. In the evacuation model, each node's weight can consist of two values; the first value being its relative distance to an exit and the second value can be a Boolean value to determine if it is occupied by a person. Since the nodes are connected by arcs, weights can also be allocated to each arc to represent their usage by the occupants. For example, arcs linking nodes near an exit can have a higher value than the arcs where few people are likely traverse. A cut can be made at the arcs which have the lowest value, indicating that less people will use those arcs and hence less communication will occur.

2.7.1.2 *Dynamic repartitioning*

Even if a good initial partitioning can be achieved, dynamical applications which involve data migration can cause the system to become imbalanced during the simulation [118, 130, 131]. For example during a pedestrian evacuation, the occupants will move towards the exits and the processors which cater for these exits will continuously be kept busy while the processors which do not serve an exit will become idle sooner. A possible solution is to perform a dynamic repartitioning of the domain. Both ParMETIS [123] and JOSTLE [121] can dynamically repartition a domain efficiently. However, it was found that the dynamic repartitioning technique can be hard to program and can be computationally expensive to repartition the whole domain again [117, 132]. For each problem case, an investigation must be made as to whether the computational expense associated with dynamic repartitioning is worse than letting the system developing load imbalance, with some processors becoming idle (and hence deflecting the goal of the parallelisation).

In the parallel computing world, it is widely known that an efficient use of parallel computing is maintaining a load balanced problem whilst minimising communication costs [121, 129, 133 - 137]. It is critical to balance those two constraints in order to maintain the efficiency of the parallel system. Computational processes that vary over time in an unpredictible way are the ideal candidates for dynamic repartitioning. A means must be devised in order to measure whether it is worthwhile to repartition a problem. A decision must be reached to ascertain which criteria is the prominent factor affecting the parallel system; is it worthwhile to allow some load imbalance if it reduces communication costs? Or is it more beneficial to having an increased communication costs whilst promoting load balance? These decisions are highly specific to the current problem and the parallel architecture. A method to determine when it is worthwhile to repartition must be devised in order to verify that the cost of load rebalancing is not outweighing the cost of running the system in an imbalanced state. Most methods either use the workload associated with the problems [130] or the run-time measurements to determine this criterion [98]. Cheng [126] uses a load function of the particles and of the mesh as a means to decide whether to warrant a repartitioning.

Once the decision to repartition has been reached, an efficient method to rebalance the load must be employed. Many algorithms use a version of the Kernighan and Lin (KL) algorithm to select

objects to transfer [132]. Depending on the applications, several variations of the KL based selection strategies are possible in measuring the gain achievable. For some applications, keeping the original partition is better than completely reshaping the domain as this involves less data migration and for others, it might be more profitable in completely repartition the whole computational domain if data transfer is not a crippling cost.

#### 2.7.1.2.1 Review of "Parallel load balancing for dynamic execution environments".

Minyard and Kallinderis [138] developed a parallel load balancing technique by using an octree-based method to calculate the amount of imbalance and to determine a new partitioning for a hybrid mesh. Similarly to what is being considered for this current research, the computational domain is divided into the number of processors which independently perform computations on their respective domains. To achieve optimal efficiency, the partitioning algorithm must be able to work effectively on a variety of geometries to partition the computational domains equally so that they have roughly the same loads. The initial grid partitioning generates sub-domains with approximately the same number of cells within them. However, imbalances in the load of a parallel environment can arise from local mesh adaptation as well as from load changes on the parallel machine itself. The use of hybrid grids (such as prisms and tetrahedra) in a computational domain involves different levels of computations and memory requirements for the different types. The computation time in a parallel solver differs between a tetrahedra and a prism. These are achieved by enabling the load balancer to take these factors into account by applying weighing values on the different types of cells. This strategy can be similarly employed in our case since nodes in the model hold different potential values based on their locations in respect to the exits. Nodes closest to the exits hold lower potential values and these nodes usually involve more congestion and hence more computations needed on them. Hence, a dynamic load balancer must be capable of adapting to changes in the mesh adaptation and the parallel architecture.

The authors state that there are two ways for the load balancer to determine load imbalance:

1. Count the number of grid cells and redistributes them evenly whilst taking account of any weight factors.

2. If there are any variation in the speeds of the processors, then the load balancer can use run-time measurements of the parallel solver to balance out the load.

If a load imbalance was observed, the dynamic repartitioning strategy follows. Most algorithms for dynamic load balancing uses two main approaches:

1. Complete repartition of the computational domain; according to the authors, this method may be fast but does not yield high quality partitions for complex geometries.

2. Local migration techniques; this technique does not have direct control over the shape and form of the partitions and large amount of data can be transferred thus incurring increased communication costs.

The same initial partitioning strategy is used for the parallel load balancer, the partition boundaries might change only slightly, hence reducing the amount of data to transfer between the processors.

The same algorithm is used on every processor, thus enabling them to know on which processor reside the new repartitioned domains and thus which cells need to be transferred. When the cells are designated to which processor they belong to, a global communication pattern for migration is generated. A global gather step is performed so that each processor knows to send data to which processors. A pack data structure is used, where all the atttributes are packed in one array so that only one communication is performed. The MPI communication protocol is used to allow this global communication step. These strategies are very appealing as the method to adopt for this current work.

The performance of this load balancer is assessed by measuring the computational time needed to rebalance the load together with the comparison of the time required for one time step prior to the rebalancing. The methods used by the authors are good candidates to be used in order to build a criterion to decide whether to repartition at the next steps.

## 2.8 Concluding remarks

An overview of the potential methods and techniques researched in this chapter is collated. The evacuation software buildingEXODUS is a behavioural model that represents its enclosure and population microscopically. Parallel processing techniques can be applied to this software to improve run-times and its ability to load large-scale problems. The domain decomposition technique with the possibility of employing a dynamic repartitioning method is a promising avenue to consider for this research. The reason is that an evacuation model contains a spatial domain which is fitting for domain decomposition. Hence, a dynamic repartitioning technique will monitor any load imbalance that can occur due to the unpredicted evacuation routes as well as any unforeseen scenarios. A distributed computer cluster with the master-slave approach is favoured as the parallel architecture to adopt due to the developers' requirement 5 from Section 1.2. The widely used MPI interface will be the chosen communication package to employ as discussed in Section 2.6.1.5. Figure 2.7 illustrates an overview of the possible strategy for this research.

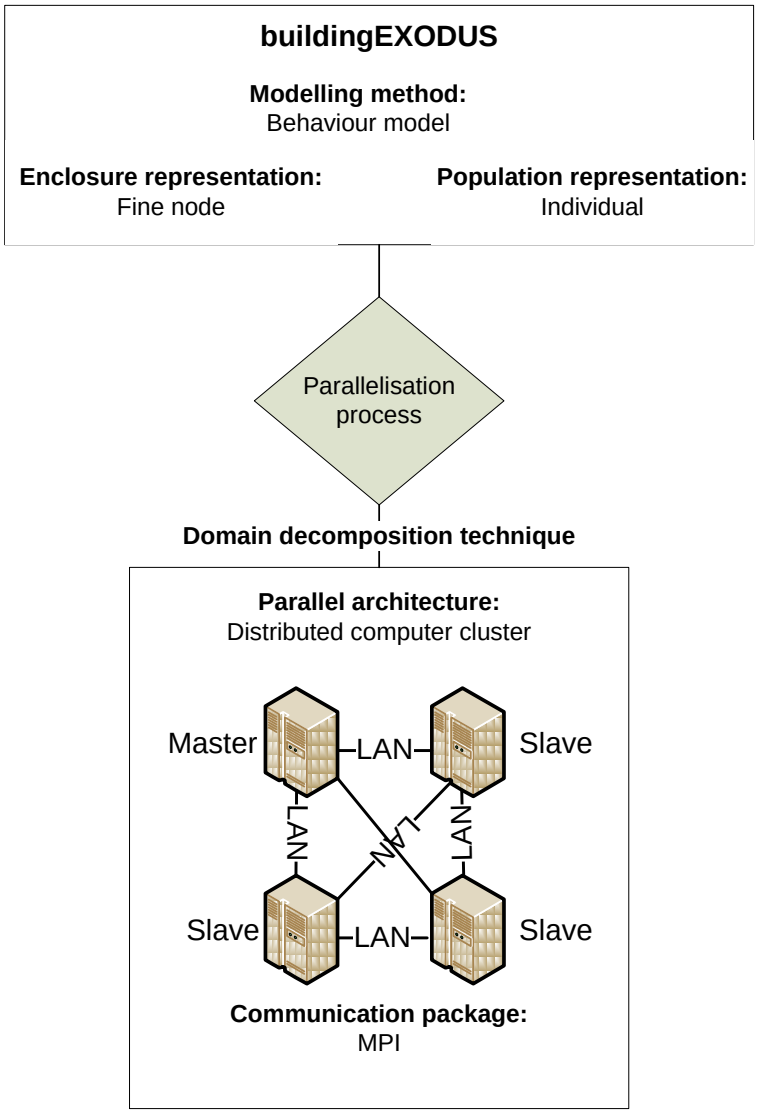



**Figure 2.7 – Overview of the possible options to consider for this research**

# 3 buildingEXODUS EVACUATION MODEL

## *3.1 EXODUS overview*

EXODUS is a suite of software tools, designed to simulate the evacuation of people from various enclosures [30]. A family of derived softwares, namely buildingEXODUS [30], maritimeEXODUS [139, 140], airEXODUS [141, 142] and railEXODUS [143] are part of the same software family. EXODUS is written in C++ and uses object oriented techniques [30] together with rule-based technology to control the simulation [16]. Thus, the behaviour and movement of people are determined by a set of rules or heuristics. For additional flexibility these rules have been categorised into five interacting sub-models, the OCCUPANT, MOVEMENT, BEHAVIOUR, TOXICITY and HAZARD sub-models. These sub-models operate on a region of space defined by the GEOMETRY of the enclosure. The interaction of these sub-models is illustrated in Figure 3.1.

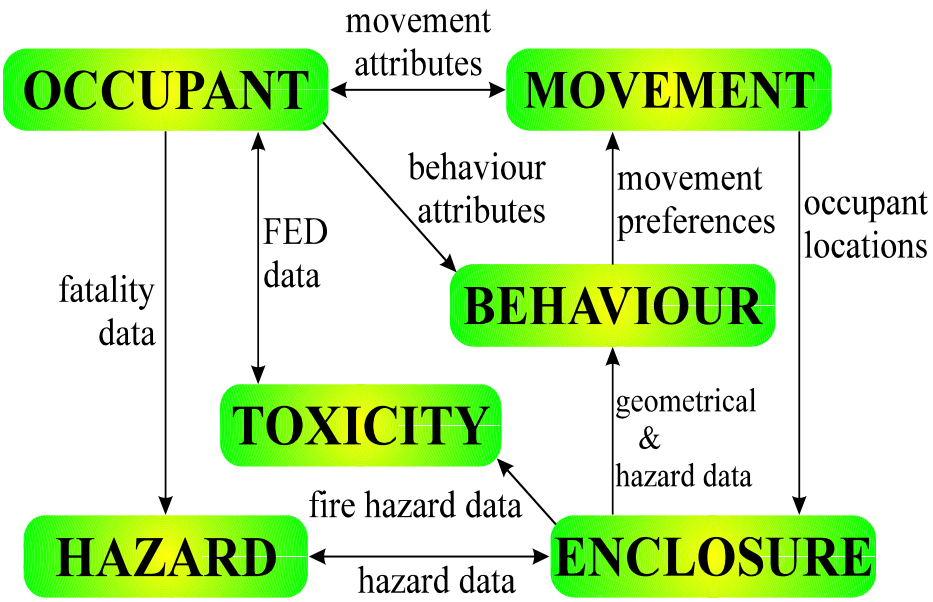



**Figure 3.1 – Interaction of the EXODUS sub-models**

Since this research is based on the derived software buildingEXODUS, a detailed review will be done on this aspect of EXODUS.

## 3.2 ***buildingEXODUS*** *overview*

### *3.2.1 Sub-models*

buildingEXODUS simulates the evacuation of people from the built environment and large open areas and the geometric representation is achieved via a 2-D spatial grid. The building layout can either be imported as a DXF file created from a CAD package or constructed by using the geometry creation feature in buildingEXODUS. The spatial grid maps out the layout of the area, defining exits, internal compartments, staircases, obstacles and any other physical aspects of the geometry. This grid consists of nodes that can only hold one person on it. The nodes are connected by arcs and the movement of people occurs along the arcs.

The temporal discretisation is ruled by a simulation clock, which occurs at every twelfth of a simulation second and this is termed as a tick. At each simulation clock, the movement of the population is calculated and performed. At every odd tick, the movement of the population is determined and at each even tick movement actually occurs [30].

The occupant sub-model caters for the nature of the population which consists of a range of people with different abilities and capabilities. Each person has various attributes, such as age, gender, weight etc, associated to them. Their movement are determined by their physical attributes as well as their knowledge about their surrounding.

The Behaviour submodel determines an occupant's responses to the current situation from their personal attributes and passes its decision on to the movement sub-model. The Behaviour submodel functions on two levels, Global and Local. Global behaviour involves implementing an escape strategy that may lead an occupant to exit via their nearest serviceable exit or most familiar exit. The desired global behaviour is set by the user, but may be modified or overridden through the dictates of local behaviour, which includes such considerations as determining the occupant's initial response, conflict resolution, overtaking, etc. In addition a number of localised

decision-making processes are available to each individual according to the conditions in which they find themselves and the information available to them. This includes the ability to customise their egress route according to the levels of congestion around them, the environmental conditions and the social relationships within the population. As certain behaviour rules, such as conflict resolution, are probabilistic in nature, the model will not produce identical results if a simulation is repeated.

## 3.3 Limitations of ***buildingEXODUS***

buildingEXODUS provides faster than real time results if the case involved is relatively small (< 10,000 population) . However, as the size of the geometry and the population size increase, the simulation will take longer to run. But as the problem size (geometry or/and population or/and behaviours represented) increases, the run time increases accordingly and can surpass the simulation time. A good example to illustrate this statement is given in Benchmark Test case 1 in Section 3.3.1.1 and in Benchmark Test case 2 in Section 3.3.1.2. Benchmark Test case 2 is a bigger version of Benchmark Test case 1 and shows how the run time can exceed the simulation time. In the case of using the software in a live incident, running the simulation in buildingEXODUS to obtain live advice and planning, will be of no use to an incident commander should run-time exceeds the simulation time. The run time should be much faster than the simulation time to enable different scenarios to be analysed and to find the best possible outcome to an evacuation.

Depending on the specifications of the CPU used to run the simulation, there will be a point where the problem size will be too large for the case to be loaded up. In that case, buildingEXODUS will fail due to memory limitations on the computer whilst attempting to load up the large case.

To get an idea of the performance and limitations of buildingEXODUS when simulating various sized problems, some benchmark values must be acquired.

### *3.3.1 Benchmark Values*

The benchmarks values to assess the performance of buildingEXODUS in terms of timings and its limitations were obtained by running five test cases with various problem sizes. All the tests were run on two processors of different specifications to illustrate how the results obtained are highly dependent on the specification of the processor used.

Another factor responsible for elevated run-times is the use of graphical visualisation whilst the software is running. When the graphics are switched off, buildingEXODUS performs faster than when the visualisation is switched on. Hence, the benchmark values include the results obtained from running with and without visualisation modes.

The processors used for the tests are shown below:

***Operating System: Windows XP Professional (5.1, Build 2600) Service Pack 2***
***Processor: Intel(R) Pentium(R) D CPU 3.40GHz (2 CPUs)***
***Memory: 3,326MB RAM***

***Operating System: Windows XP Professional x64 Edition (5.2, Build 3790) Service***
***Processor: Intel(R) Pentium(R) III Xeon processor (2 CPUs), ~3.2GHz***
***Memory: 8,110MB RAM***

#### 3.3.1.1 *Benchmark test case 1 - 150m x 150m enclosure populated by 10,000 occupants*

The first case is designed to accommodate a population of 10,000 people occupying a simple square enclosure. Since the size of the population is quite large the geometry will be assumed to be of an open ground assembly type (e.g. pop and rock concert area). This was approximated to 150m x 150m. The exits were placed uniformly around the perimeter of the square. Figure 3.2 shows a crude diagram of the geometry, with some additional details, of the enclosure used.

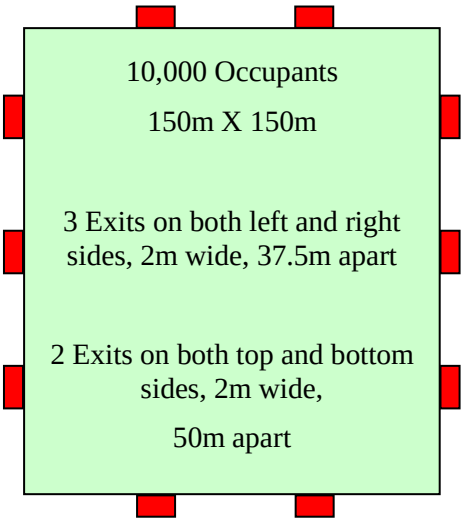



**Figure 3.2 – Square geometry measuring 150m by 150m.**

An average of the simulation and wall-clock times obtained from running the simulations of evacuating 10,000 occupants are displayed in Table 3.1.

**Table 3.1 – Times in seconds for Benchmark test case 1**

| | **Graphics ON** | | **Graphics OFF** | |
|---|---|---|---|---|
| | **PC 1** | **PC 2** | **PC 1** | **PC 2** |
| Simulation Time (s) | 277.00 | 273.00 | 284.33 | 274.67 |
| Wall-clock Time (s) | 175.67 | 72.67 | 127.00 | 38.08 |

Benchmark test case 1 is a sufficiently small problem size to attain faster run-times than simulation times. For both the scenarios of having different specifications for the processors as well as running the problem with the graphics switched on and then off, all the wall-clock times obtained are less than their associated simulation time advised by the software. Running Benchmark test case 1 on PC 1 returns a wall-clock time improvement over its associated simulation time of 36.6% with the graphics switched on and of 55.3% when the graphics were

turned off. When PC 2 was used, a wall-clock time improvement of 73.4% can be observed when the graphics were switched on and an increase of 86.1% with no visualisation features. Running the case without visualisation returns an improvement of 13.8% for PC 1 and of 19.1% for PC 2 compared to using the visualisation features. Please note that the difference in the Simulation times obtained from using the two processors are different due to the fact that evacuation models which do not use a formula to predict the movements of its population, will give out different simulation times at every run. This is due to the unpredictibility of the population's movement that the model promote.

3.3.1.2 *Benchmark test case 2 - 245m x 245m enclosure populated by 30,000 occupants*

The second case is based on the same principles as the first case but designed to accommodate a population of 30,000 people. Seven exits were uniformly placed across the perimeter of the geometry as shown in Figure 3.3.

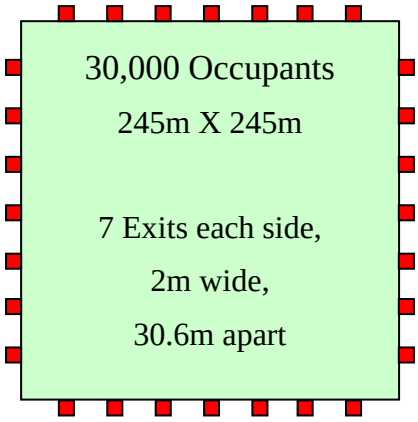



**Figure 3.3 – Square geometry measuring 245m by 245m**

The results obtained from running 'Benchmark test case 2 - 245m x 245m enclosure populated by 30,000 occupants' is displayed in Table 3.2. All 30,000 occupants evacuated the geometry in the averaged simulation times displayed in Table 3.2.

**Table 3.2 – Times in seconds for Benchmark test case 2**

| | **Graphics ON** | | **Graphics OFF** | |
|---|---|---|---|---|
| | **PC 1** | **PC 2** | **PC 1** | **PC 2** |
| Simulation Time (s) | 378.33 | 375.50 | 377.00 | 374.50 |
| Wall-clock Time (s) | 758.33 | 307.50 | 608.67 | 234.00 |

Benchmark test case 2 is a larger version of Benchmark test case 1 both in terms of geometry and population. In this instance, running the tests on PC 1 produced higher wall-clock times than their associated simulation times, whether the graphics were switched on or off. A possible reason for incurring higher wall-clock times than their corresponding simulation times is the extra calculations involved in moving a larger population within a larger geometry. With respect to their associated simulation times, the wall-clock times were more than 100% slower with the visualisation features and 61.4% slower with no graphics. However, running the same test case on PC 2 incurred a reduced wall-clock times compared to their associated simulation times, which are improvements of 18.1% and 37.5% with and without graphics respectively. Expectedly, switching the graphics off incurred less time than when the visualisation features were used, specifically 19.7% and 23.9% less time for PC 1 and PC 2 respectively.

3.3.1.3 *Benchmark test case 3 - Multi-storey building populated by 8,120 occupants*

Case 3 involves a multi-storey building with 8 main exits. The building consists of 50 floors including the ground floor. The population size was 8,120 occupants. The results obtained from testing Case 3 in simulating the evacuation of all 8,120 occupants are displayed in Table 3.3.

**Table 3.3 – Times in seconds for Benchmark test case 3**

| | Graphics ON | | Graphics OFF | |
|---|---|---|---|---|
| | **PC 1** | **PC 2** | **PC 1** | **PC 2** |
| Simulation Time (s) | 2516.00 | 2524.00 | 2491.50 | 2513.50 |
| Wall-clock Time (s) | 1472.50 | 782.50 | 1254.50 | 719.50 |

This case comprises of a big geometry but with a smaller population than previously tested. All the scenarios tested produced better wall-clock times than their simulation times counterparts. PC 1 permitted a progress over the simulation time of 41.5% with the graphics switched on and of 49.7% with the graphics switched off. Similarly, running the test case on PC 2 provided an improvement over the simulation times by 69% and 71.4% with the graphics switched on and off respectively. Unsurprisingly, running the test cases without any visualisation reduces the times from running them with the graphics switched on. Namely, a 14.8% decrease for PC 1 and an 8% decrease when PC 2 was used.

3.3.1.4 *Benchmark test case 4 - Creation of a 750m x 1,000m enclosure*

Benchmark test case 4 is the attempted creation of a rectangular geometry measuring 750m by 1,000m on PC 2. After about 18 hours of building the nodes and arcs to create that geometry, the processor ran out of memory and the building process failed.

3.3.1.5 *Benchmark test case 5 – Real geometry: World Trade Centre's north tower*

A good example of the application of buildingEXODUS on a real geometry is the simulation the evacuation of the World Trade centre (WTC) [144, 145]. The simulations attempted to reproduce the evacuation of the North Tower of the World Trade Centre. This involved a building of 110 storeys and two different target populations, one involving approximately 8,200 occupants and the other involving 21,000 occupants. With a population of 8,200, the actual evacuation may require around 100 minutes in real time while with a population of 21,000 the evacuation may require approximately 150 minutes. Using a 3.6 GHz Pentium 4 PC with 3.25 GB RAM these

simulations required approximately 25 minutes and 120 minutes respectively. Even at four times faster than real time, the 25 minute simulation of the WTC evacuation is not sufficiently fast to be of use to incident commanders [16].

#### 3.3.1.6 *Overview*

The benchmark test cases examined produced diverse results: from having faster run-times than simulation times, slower run-times or even the inability to load up a large-scale problem. Even when the run-times were faster than the simulation times, the gain achieved was not significantly large enough to be of any use to incident commanders.

A possible remedy to improving the run-times as well as allowing the loading up of large-scale cases is the expensive purchase of more powerful processors. However, due to the software's developpers' requirement 5 from Section 1.2, this option is not viable. The other possible solution to the problems posed above is the use of parallel processing.

## *3.4 Parallel computing as a solution to the limitations of buildingEXODUS*

Given the issues of speed as stated in this chapter, parallel processing is a method that can be used to reduce run-times of very large cases that would have been impossible on a single processor. This method typically distributes the workload amongst a number of CPUs. A parallel implementation of the buildingEXODUS software will distribute an evacuation scenario across a number of computers making use of the available memory and processing capabilities of the computers in the cluster or network.

There are two main benefits associated with this approach. Firstly larger scenarios can be modelled than would be possible with a standard evacuation model. As each CPU will now hold only part of the problem size, this allows the simulation of very large building complexes or potentially even large urban spaces to be modelled in far greater detail than was previously possible. Secondly large scenarios can run more quickly than was previously possible and potentially much faster than real time. Instead of one CPU catering for the simulation of the

whole problem size, parallel processing will enable each CPU to now simulate a fraction of the original load, which technically should reduce the run times.

# 4 PARALLEL IMPLEMENTATION OF buildingEXODUS

The buildingEXODUS code is complex with over half a million lines of codes, so a simplified prototype version miniEXODUS was written to function similarly to buildingEXODUS to test the feasibility of the parallel approach. This miniEXODUS was used to test a number of strategies that then fed into the developement of parallel buildingEXODUS as carried out by Grandison et al. [16]. miniEXODUS was used by both the software developers and in this current research to extensively test the parallel approaches needed for the final parallel implementation of buildingEXODUS. However, the source code necessary for the partitioning strategies (Chapters 5 and 6) devised in this research were accessible for the software modifications. Consequently in this chapter, there are some collaborated materials from the software developers (Sections 4.2.4.1.1, 4.2.5, 4.2.5.1, 4.2.5.2, 4.2.5.3, 4.2.5.4, 4.2.5.5, 4.2.6.1, and 4.3.1) which are briefly explained to aid the understanding with respect to the research undertaken by the author. These collaborated materials can be found in more detail in the Appendix.

## *4.1 Prototype: miniEXODUS*

The miniEXODUS prototype performs the very basic functions of buildingEXODUS, namely simulating the evacuation of people from a geometry. Similar to how buildingEXODUS allows a person to move, the movement of the individuals are performed by having the node attracting a person onto it rather than a person choosing which node to move to. miniEXODUS calculates the movement of people by the Potential Route Map, which makes people traverse to nodes which are closest to an exit. buildingEXODUS has many methods to simulate the evacuation of people and the Potential Route Map technique is one of them. In the first instance, to parallelise buildingEXODUS directly would have been too demanding in terms of sifting through all the code and the interacting sub-models. Hence, miniEXODUS was used to assess a number of parallelisation strategies and techniques so as to select the best method to parallelise buildingEXODUS.

### *4.1.1 Parallelisation strategies used to parallelise miniEXODUS*

From the initial research into which parallel strategy seems most beneficial for an evacuation model (Section 2.6.1.4), the domain decomposition technique was adopted in the parallelisation of miniEXODUS. The physical domain of the problem was decomposed by using one of the different partitioning strategies investigated; See Section 4.1.1.1. Using this decomposition technique resulted in the creation of halo nodes to allow the communications of the individuals at the boundaries as well as the synchronisation amongst the processors during the simulation. The investigation into the creation of halo nodes gave an important insight on the size of the halo regions. The halo regions on each processor need to be at least 2 nodes wide to permit the movement of the people. This is due to the fact the movement of an individual is determined by the neighbouring nodes surrounding the current node which the occupant is on. An exploration into how the synchronisations amongst the processors were executed during each clock-tick was performed so that each processor is on the same simulation time. The communications processes had to be carefully co-ordinated to allow the smooth transmission of the data. It is not the case that a processor can simply send and receive messages to each other. Hence, the communication strategies were devised and explored so that the occupants were properly communicated across, such as a copy of an occupant on a halo node needs to be created on its corresponding halo node on the neighbouring processor and then deleted as soon as that occupant moves out of the boundary region.

Significant insights into how the parallel evacuation model will perform were gained with the parallelisation of miniEXODUS. A positive observation was that the data communications were very fast and did not appear to be a limiting factor in the time taken to perform this operation. Initially the parallelised version was tested on two processors and the strategy was working. When the number of processors was increased, another crucial discovery was that the parallel model cannot cater for multi-connected partitions. A partition having more than two neighbouring partitions creates a copy of the same nodes in each halo region created at the boundaries. For example, a node (e.g. node X) can have several neighbouring processors. Please refer to Figure 4.1 for the illustration of node X denoted by a star. In those cases, different sub-domains are associated to that particular node X; this leads to the problem of having multiple copies of a single node and furthermore determining which region is responsible to move an occupant of that

node, as in buildingEXODUS occupants get moved by a neighbouring node rather than the node it's currently occupying. These types of nodes create overlapping halos.

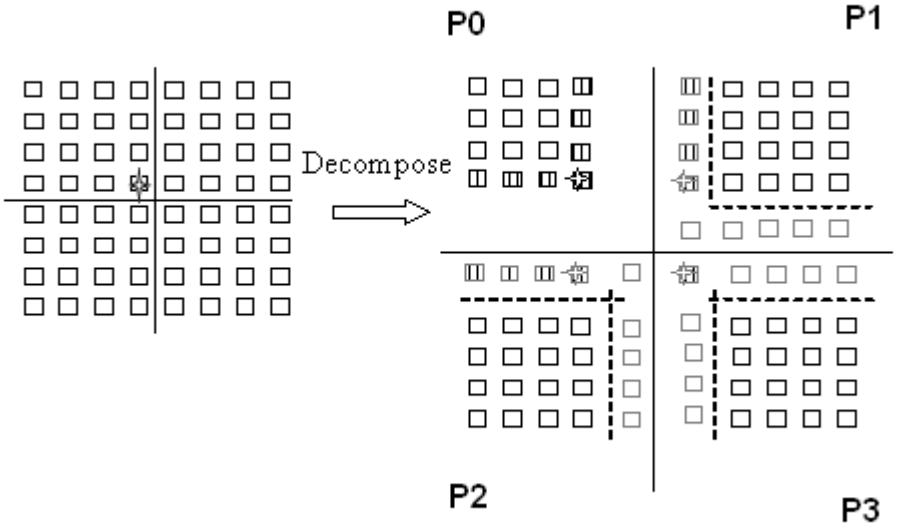



**Figure 4.1 - Node X virtually exists on three other domains on different processors as a multi-halo node**

The problem is that node X resides on four different processors and hence can attract a person onto it so that the same person is now being simulated by four processors. This created the problems of having multiple copies of a same person being simulated by more than one processor. This problem was not researched further in the parallel version of miniEXODUS as by then the parallel strategies were already implemented in buildingEXODUS and the software developer had already found a quick solution to that problem by creating Multi-Halo nodes as explained in Section 4.2.4.1.1.

#### 4.1.1.1 *Partitioning strategies attempted*

In an attempt to investigate the impact of having different partitioning strategies with different population loads on the parallel performance, several test cases were run. Test Case 1 is one of the test cases attempted.

**Test Case 1 - Geometry modelled on the Beijing High Street.**

This geometry is roughly rectangular in shape, containing 132,878 nodes and 9 exits located around the geometry.

This particular geometry was chosen as one of the main uses of the parallelised version of buildingEXODUS is to be used on large open-spaced and non-built geometries. Different population loads, namely 5,000, and 15,000 people were simulated in an attempt to test the robustness and communication costs of the system as well as how they impact on the speedups obtainable. Two identical processors[1] available at that time were used to run those tests.

Three main partitioning strategies were tested for Test Case 1, the Equal Partitioning method (where the geometry was equally divided into 2, 4 and 9 dub-domains), the Potential Route Map (PRM) Method (automatically generating 9 sub-domains from having 9 exits, please see Section 5.1.4.1 for more information about this method) and the partitioning software METIS (used to partition the domain into 2 and 9 sub-domains so as to get a direct comparison to the Equal Partitioning and the PRM methods).

The result from using these partitioning strategies on Test Case 1 populated by 5,000 people is displayed in Figure 4.2.

---

[1] *Processor: Intel(R) Pentium(4) 1.5GHz*
*Memory: 60GB RAM*

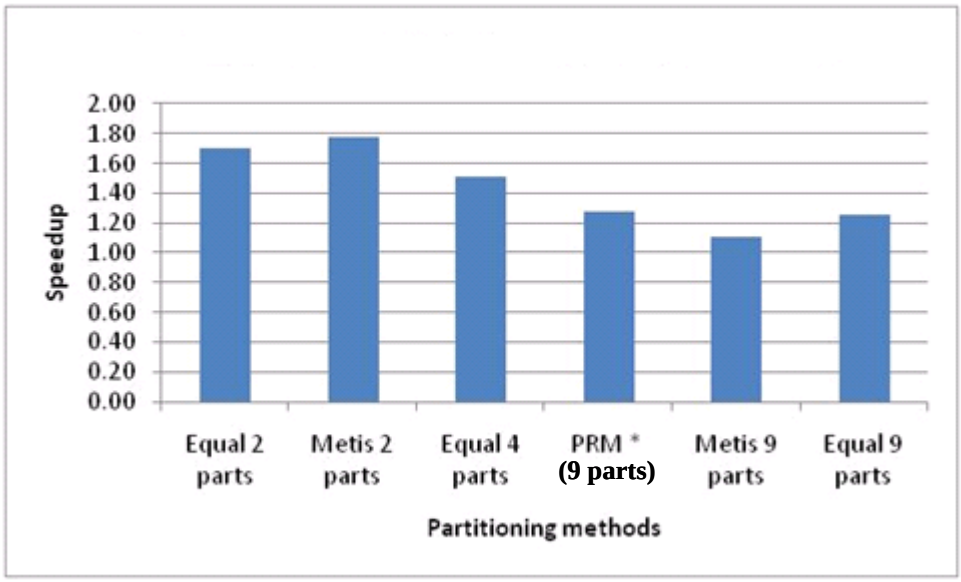



**Figure 4.2- Speedup values obtained for using Parallel miniEXODUS with different partitioning strategies on Test Case 1 populated by 5,000 people**

From these results, the parallelisation of the miniEXODUS prototype can be considered a success, as speedup values greater than one were obtained. The best and worst speedup values were when the partitioning software METIS was used to partition the geometry into two and nine sub-domains respectively. The chief deduction is that the parallelisation techniques applied is a success and can be ported on buildingEXODUS. Also, it is found that the choice of a particular partitioning method is crucial in obtaining good speedup values.

Increasing the population load to 15,000 people in the same geometry whilst using the same partitioning methods used in the previous test run with 5,000 people, produced the results displayed in Figure 4.3.

[1] * *PRM: Potential Route Map*

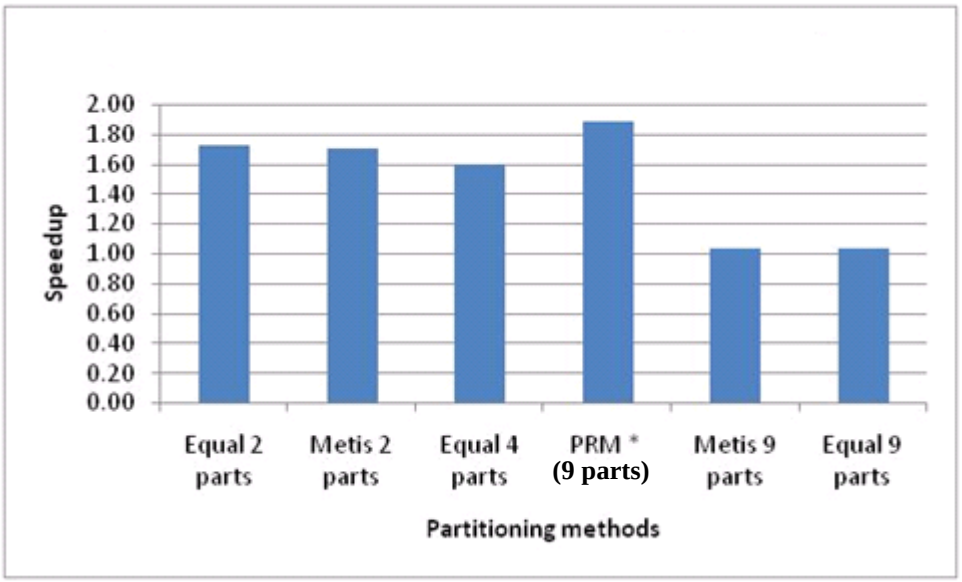



**Figure 4.3 - Speedup values obtained for using Parallel miniEXODUS with different partitioning strategies on Test Case 1 populated by 15,000 people**

Some interesting observations can be made from the results displayed in Figure 4.3. Here again, all the speedup values obtained are greater than one, suggesting that the parallelisation of the prototype is a success and can achieve faster times than using only one processor to run the test case. However, increasing the population load seems to have a negative impact of using METIS (9 sub-domains) and the Equal Partitioning (9 sub-domains) methods. The most likely cause degrading the results of these two methods is the extra communication costs of the increased population load. The PRM method also has nine sub-domains, but because the partition boundaries are strategically placed so that fewer people would be crossing these boundaries, this method produced the best result with a speedup of more than 1.8. The importance of choosing the right partitioning strategy seems to be the dominating factor in achieving good speedups.

### 4.1.2 Remarks

Various important insights such as how to communicate the people at boundaries, synchronisation issues and the problems encountered with having multi-connected halo regions were gained so as to prepare for the actual parallelisation of the buildingEXODUS software. This prototype also acted as a feasibility test to decide whether to continue with the parallelisation of buildingEXODUS. From the results obtained from running the test cases, the feasibility tests showed there is a strong potential to provide good speedup values in having a parallelised version of buildingEXODUS. However, a good choice of the partitioning strategy is necessary in order to get good results, hence more research in finding this optimal partitioning strategy is necessary.

## 4.2 Parallelisation Strategies employed for buildingEXODUS

Having carried out initial investigations from miniEXODUS, the successful parallelisation strategies were ported to the parallelised buildingEXODUS and were further developed. However due to the complexity of buildingEXODUS, a more detailed investigations into the parallelisation methods must be conducted since all avenues (such as Multi-connected regions, output files and any other aspects of the software that miniEXODUS did not model) could not be explored by the parallelised miniEXODUS.

### 4.2.1 At what point does parallelisation become a benefit over the serial algorithm

A parallel system of processors is generally considered a benefit over a single processor as the combined power of a parallel configuration is capable of solving problems faster than a single processor. However, configuring a parallel system incurs parallelisation costs such as communication and synchronisation costs that a single processor does not incur. Generally if the parallelisation costs is minimal compared to the algorithmic costs of a problem, a parallel system will fare better than running that problem on a single processor. However, if the parallel costs are higher than the computation costs, then no advantages can be gained from using the parallel system. A bigger problem size generally incurs more computations relative to the parallel costs

than a smaller case. Hence, the benefit of employing a parallel system is dependent on the problem size.

Two tests were run on two processors with specifications[1] shown below to determine at what point a parallel system becomes advantageous over a single processor. A scenario was applied to gradually increasing the population size to determine the point when a single processor is no longer beneficial to use, that is when the speedup is greater than one.

**Test 1:** Geometry measuring 700 x 500 nodes, with 24 exits located at regular intervals along the perimeter of the geometry.

**Test 2:** Geometry measuring 50 x 250 nodes, with one exit along one end of the geometry.

The results are shown in Table 4.1:

**Table 4.1 - Speedup values obtained to test when parallelisation becomes beneficial**

| Population size | 1000 | 2000 | 3000 | 4000 | 5000 | 6000 |
|---|---|---|---|---|---|---|
| **Speedup (Test 1)** | 0.76 | 0.98 | 1.12 | 1.25 | 1.31 | 1.39 |
| **Speedup (Test 2)** | 0.38 | 0.89 | 1.47 | 1.69 | N/A | N/A |

A starting population size of 1,000 was used and the resulting speedups were 0.76 and 0.38 for case 1 and case 2 respectively, which shows a low speedup. This population size was gradually increased and when the speedup reaches more than one, it was considered that the parallel system

[1] ***Specifications of processors used***

***Operating System: Windows XP Professional x64 Edition (5.2, Build 3790) Service***

***Processor: Intel(R) Pentium(R) III Xeon processor (2 CPUs), ~3.2GHz***

***Memory: 8,110MB RAM***

was better than the single processor. It was found that on average, using 3,000 or more occupants is the point when the parallel system starts yielding positive results. According to this finding, it is not considered advantageous to repartition a geometry if there are less than 3,000 occupants, leaving this population in a single processor theoretically is better than trying to repartition the load to other processors. There is a possibility that this value of 3,000 will differ if the specifications of the processors used are different from those that were used to run the test. Due to the inaccessibility of another set of processors with different specifications to further test this finding, this value was jointly used with the LPT algorithm to determine when to repartition the geometry.

### *4.2.2 Parallel Architecture adopted*

A network of processors will be used to accommodate this parallel architecture as demonstrated in Figure 4.4. Due to requirement 5 imposed by the software developers, a master-slave approach was chosen where there is a master processor which controls all the slave processors. The master processor can work as a slave as well and do extra work. All the processors will need to communicate amongst them and the message passing interface (MPI) will be used to enable the communications.

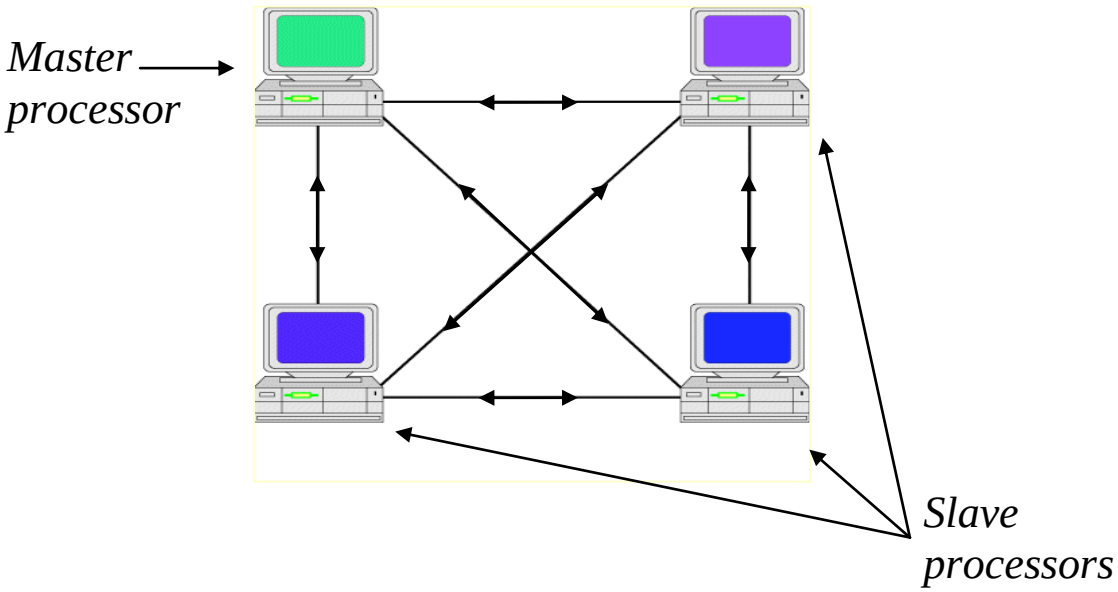



**Figure 4.4 - Parallel Architecture adopted**

### *4.2.3 Problem decomposition*

For the decomposition of an evacuation model, the two most attractive methods are either the population decomposition or the geometric decomposition.

#### 4.2.3.1 *Population decomposition*

Similarly to how Hatsky [119] used a particle decomposition strategy, a population decomposition entails dividing the whole population into sub-groups. Each sub-group resides on a processor and will remain in the care of that processor until every person from that subgroup has evacuated.

Figure 4.5 illustrates this decomposition technique. Consider a geometry with a population load (represented by the dots). At the start of the simulation, the population load is divided amongst the number of processors available and each sub-load is allocated to them (illustrated by the different coloured dots) and the whole geometry is stored on each processor. During the simulation, each processor simulates its local population load wherever they are on that domain.

Depending on the parallel architecture used, this method can be useful as explained in Section 2.5. For example, on a Single Shared memory (SM) machine, the interactions amongst the individuals is not a problem as there is no need to communicate the data required to calculate the interactions. Whereas, on a distributed memory system (DM), each sub-group residing on each processor will be stored locally. In computer terminology, this is a network/cluster of processors linked together over a Local Area Network (LAN). Hence, as large amounts of interactions amongst the occupants simulated on different processors occur, an increase in communication costs is expected.

Another shortcoming to using this method is the fact that each processor must hold the entire geometry, which will be a limiting factor as to the size of a test case, no matter how many processors are utilised. In summary, population decomposition aims at subdividing the population and assigning a fraction of that population on each processor which must hold the entire geometry.

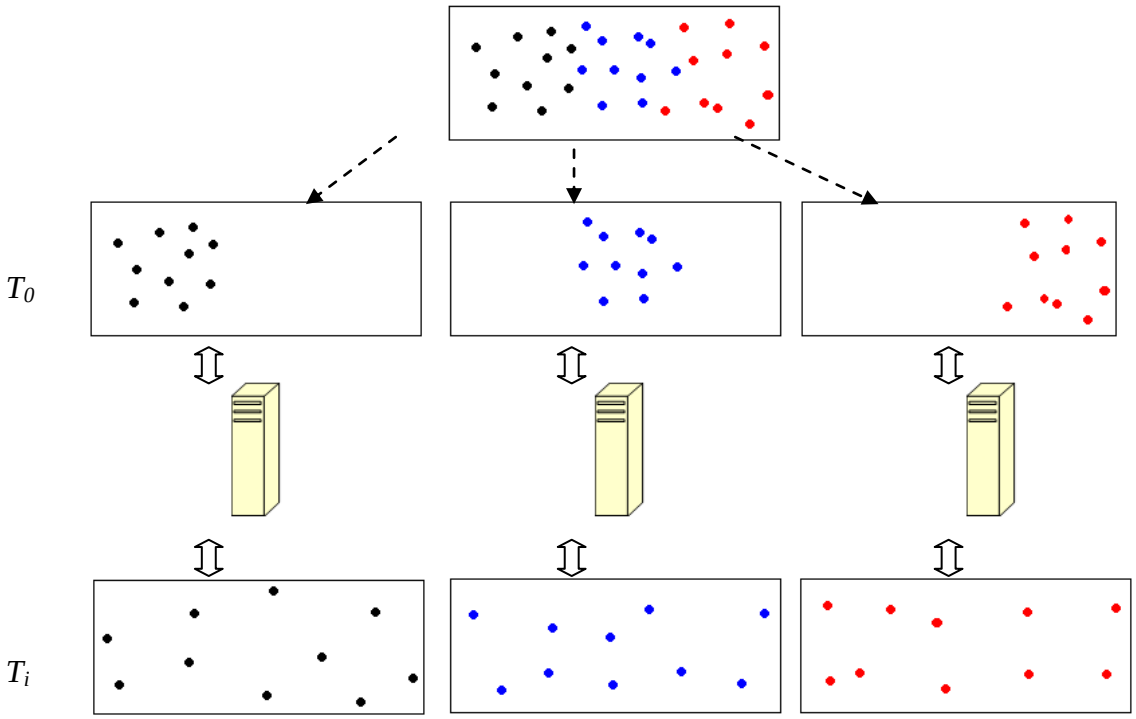



**Figure 4.5 – Population decomposition on a DM system connected via a LAN network**

#### 4.2.3.2 *Domain decomposition*

Domain decomposition (Section 2.7) is the systematic partitioning of the main geometry into a number of sub-domains. Each processor is responsible for one or more of those sub-domains as well as the occupants who happen to reside on them. Each processor will run its own copy of the evacuation modelling code on its allocated sub-domain(s) and at the boundaries of each sub-domain; communications of the occupants residing on those boundaries must be communicated to the processor responsible for the neighbouring sub-domain. So using this strategy, a particular occupant may potentially be handled by several processors.

Figure 4.6 shows a domain being partitioned into two sub-domains, each residing and being simulated on a different processor.

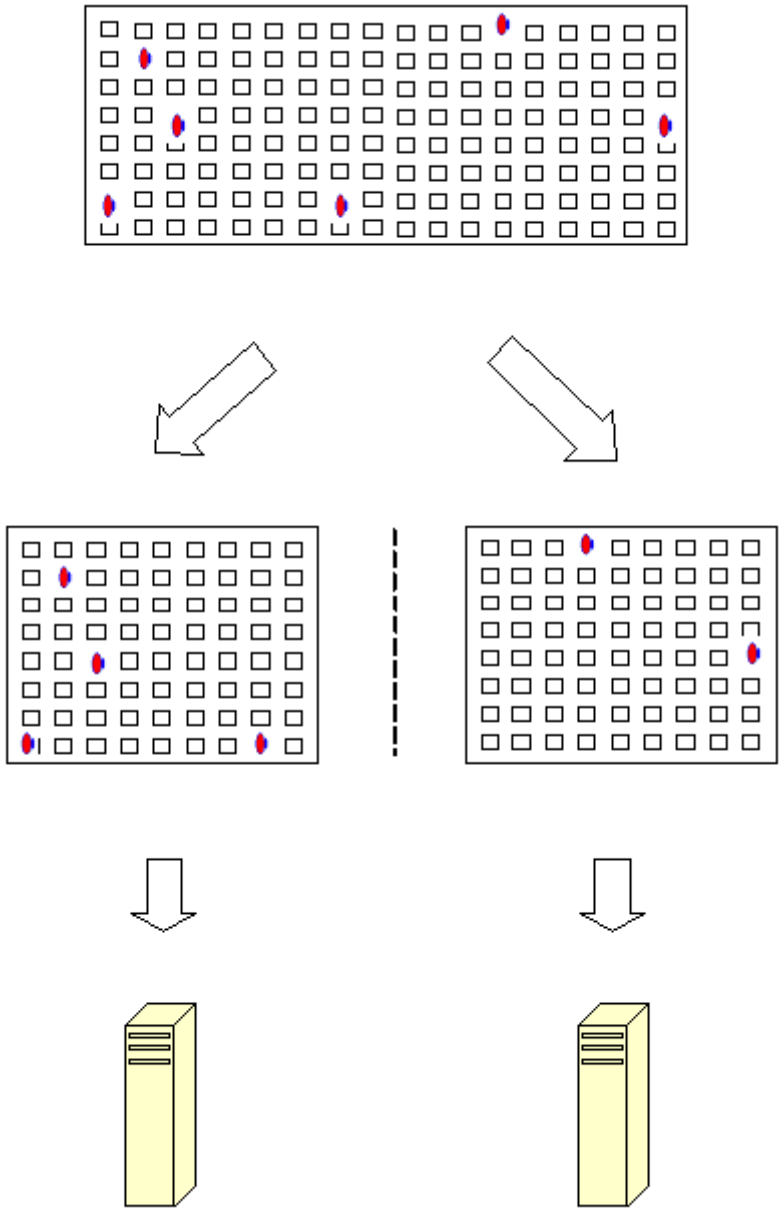


**Figure 4.6 – Domain decomposition on a DM system connected via a LAN network**

4.2.3.3 *Selected decomposition strategy*

With the drawbacks of a population decomposition technique of each processor having to hold the entire geometry and having a high communication overhead, the domain decomposition technique seems to potentially resolve those issues. Using a domain decomposition strategy, the

problem of holding the entire geometry is solved as the geometry is divided into sub-domains. Additionally each processor will be responsible for only a fraction of the population size, as only the occupants of the allocated sub-domain will be simulated on each processor. This method consequently generates a population decomposition as well since the population is being partitioned together with the domain. From the software developers' requirement 5 in Section 1.2: "Ideally the parallel architecture could be office PCs connected by a LAN network.", together with the advantages that this strategy provides, the chosen partitioning technique is the domain decomposition strategy as it favours the Distributed Memory (DM) model of parallel processing.

Also from requirement 6: "There should only be one EXODUS source code i.e. separate parallel and serial source codes would not be developed." The program was written to function on both a SM machine as well as the DM system, removing the need to have two different set of codes. To achieve this, the serial code was modified to include the parallel capabilities. The code was written using the Single Program Multiple Data (SPMD) paradigm. With this methodology only one executable is used and a copy of the executable is launched on each processor which operates on its own part of the problem domain.

This selected decomposition strategy necessitates the communication of the individuals when they are crossing over the boundaries of the partitioned domains. Section 4.2.4 portrays the methods necessary to allow the communications of the people.

### *4.2.4 Communication Strategy*

#### 4.2.4.1 *Halo regions*

Halo regions are created at the partition cuts to represent the boundary nodes on the neighbouring processors. The halo nodes are there to enable the communication amongst the neighbouring sub-domains about the occupants residing on those boundary nodes. On each sub-domain's boundaries, a set of Inner Halo nodes are identified and they will be the corresponding Outer Halo nodes on the respective neighbouring processor. An occupant residing on an Inner Halo node will be mirrored on that same corresponding Outer Halo node on the neighbouring

processor. This technique is necessary to allow the movement of the people on those boundaries as a transition between processors' sub-domains. Figure 4.7 illustrates this concept.

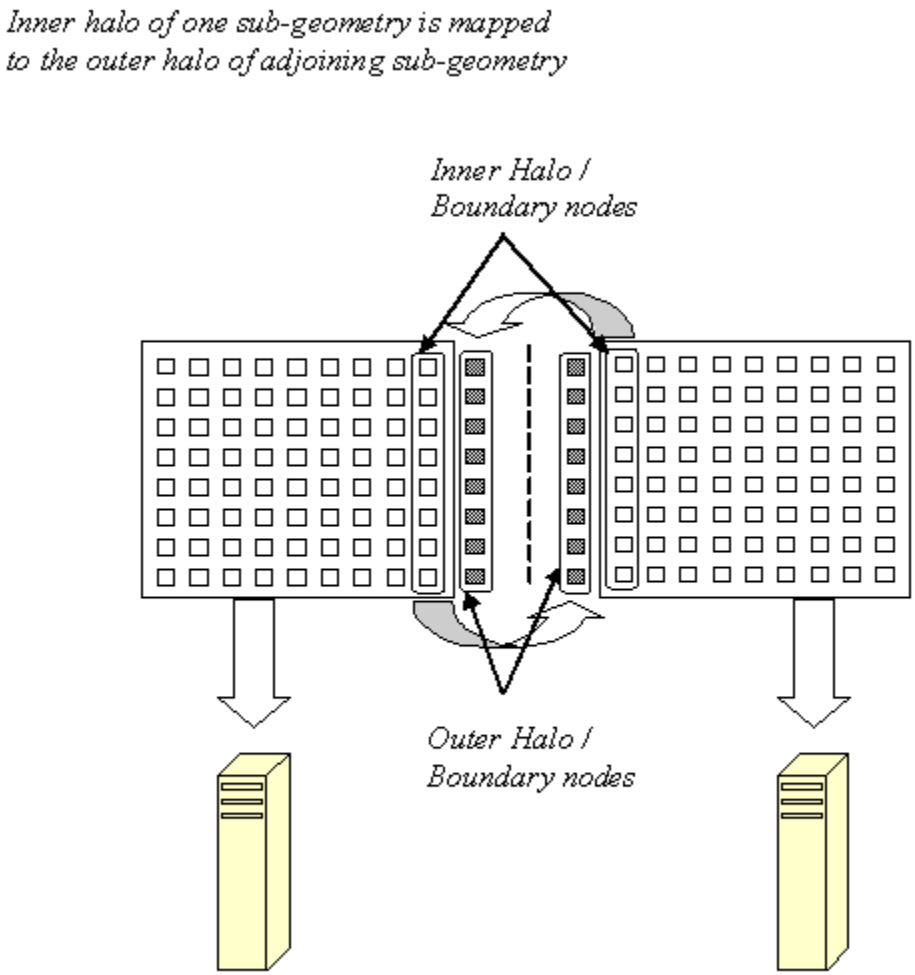



**Figure 4.7 – Halo regions representing boundary nodes**

4.2.4.1.1 Multi-halo regions

The previous example demonstrated in Figure 4.7 had a one dimensional partitioning strategy, meaning that boundary nodes have only one neighbouring processor. However it is possible to have other types of partitions where a certain node X can have several neighbouring processors. Please refer to Figure 4.8 for the illustration of node X denoted by a star. In those cases, different

sub-domains are associated to that particular node X; this leads to the problem of having multiple copies of a single node and furthermore determining which region is responsible to move an occupant of that node as in EXODUS, occupants gets moved by a neighbouring node rather than the node its currently occupying. These types of nodes create overlapping halos.

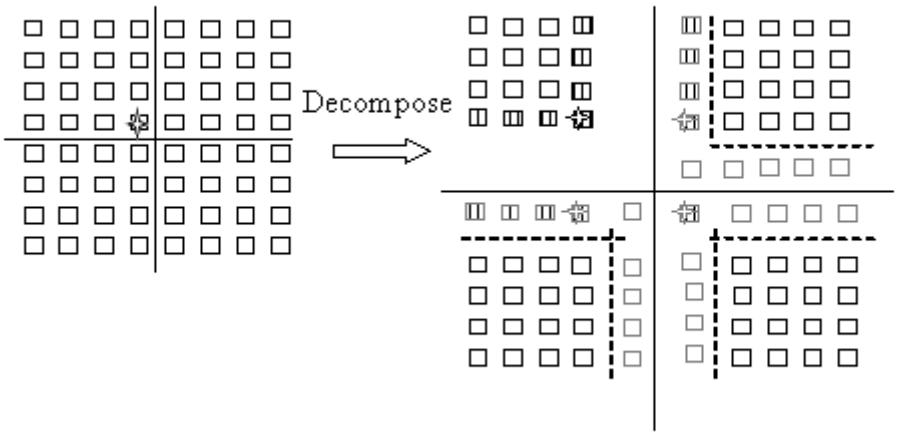



**Figure 4.8 - Node X virtually exists on three other sub-domains on different processors as a multi-halo node**

To alleviate this problem, halo nodes that share sub-domains are converted into a new sub-domain called a multi-halo region. Figure 4.9 illustrates this conversion. This multi-halo region is catered for by the master processor, similar to how Quinn et al [89] performed their communication strategy. This is a simplification of what actually occurs in the implementation. These multi-halo regions also have a required minimum width of five nodes to avoid any communication issues that may arise if the halo regions are too thin.

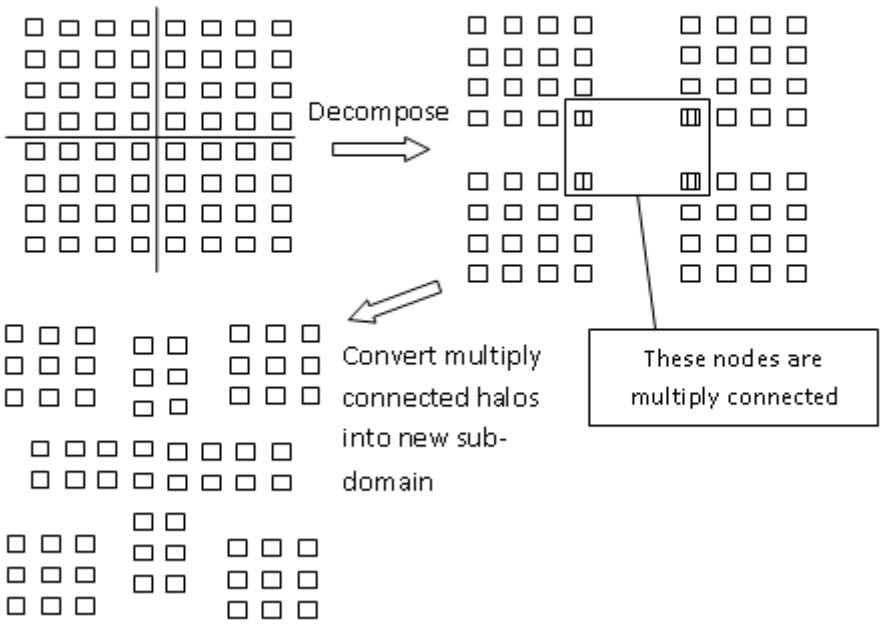



**Figure 4.9 - Conversion of multiple connected nodes into new sub-domain**

### *4.2.5 Movement of People*

The movement of people briefly presented here is included for completeness, see Appendix 11.1 and Grandison et al. [16]. These are included in this thesis to aid the understanding of the research done for this dissertation.

#### 4.2.5.1 *Person transfer and movement across sub-domains boundaries*

Figure 4.10 demonstrates the movement of the occupants on boundary areas. Consider two sub-domains, each handled by a processor. Each processor is responsible for calculating the computations for the evacuation simulation for the people located on their respective sub-domain. In the example shown in Figure 4.10, the person is moving from left to right and must be transferred from one sub-domain to the other. The movement across sub-domains can be briefly explained:

1. In Figure 4.10(a), the person is approaching the sub-domain boundary but is currently controlled by the left hand sub-domain (LHSD).

2. As the person enters the Inner Halo region of the LHSD (see Figure 4.10(b)), the LHSD now sends a message concerning the person located on the Inner Halo node to the right hand sub-domain (RHSD) and the RHSD now creates a copy of the person on its Outer halo node. At this point the person is now controlled by the RHSD as the person wishes to move onto the RHSD.

3. As the person moves onto the RHSD (see Figure 4.10(c)) this information is sent back to the LHSD which updates the person on its side to move to its Outer Halo region.

4. In Figure 4.10(d), the person now continues its journey on the RHSD and as it moves away from the boundary the LHSD is instructed to recycle its copy of the person as that copy is no longer needed.

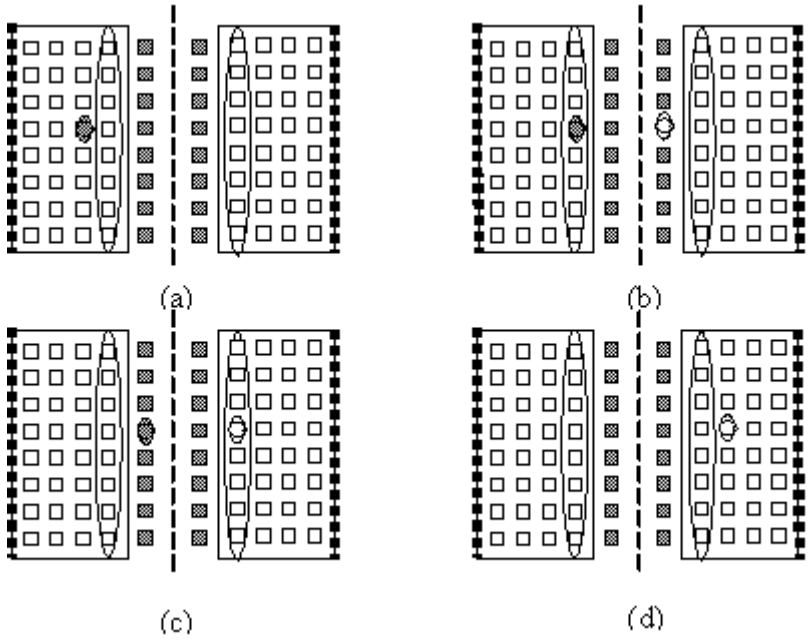



**Figure 4.10 – The movement of an individual across sub-domains in the parallel version of EXODUS [16]**

#### 4.2.5.2 *Movement Algorithm*

The overall simulation is governed by the global Simulation Clock which is a timeline division of 1/12s. On these ticks occupants decide what their future actions will be; this is generally a movement towards the nearest exit. The population is looped over until all the PETs (Personal Elapsed Time) of each of the individuals have incremented beyond the Simulation Clock which may be due to a movement to the next node or due to congestion. If congested, a decision to wait in its current position is put in force until the next tick of the global Simulation Clock. The global Simulation Clock keeps ticking until all the occupants have exited the evacuation scenario.

#### 4.2.5.3 *Serial code enhancement*

The list of individuals used for the movement algorithm was optimised. In the revised algorithm the list used for movement would be modified to only include occupants that still needed to be moved. The increase in performance was case dependent but was found to be on average 47% faster than the original algorithm with the improvement most marked when higher congestion levels existed.

Please refer to Appendix 9.2 for a detailed explanation of those changes. These improved codes were then fed into the parallel implementation to reduce the computational time of running the software.

##### 4.2.5.3.1 New Benchmark values after the serial enhancements

The same cases used to benchmark the values in 3.3.1 Benchmark Values are now demonstrated in Table 4.2 by using the improved serial codes. The numbers in *italics* are the values demonstrated in 3.3.1 Benchmark Values; they are included here to make the comparisons easier.

The processors used for the tests are shown below:

***Operating System: Windows XP Professional (5.1, Build 2600) Service Pack 2***
***Processor: Intel(R) Pentium(R) D CPU 3.40GHz (2 CPUs)***
***Memory: 3,326MB RAM***

***Operating System: Windows XP Professional x64 Edition (5.2, Build 3790) Service***
***Processor: Intel(R) Pentium(R) III Xeon processor (2 CPUs), ~3.2GHz***
***Memory: 8,110MB RAM***

**Table 4.2 - Comparison between the original serial code and the improved serial code for Benchmark test case 1**

| | **Graphics ON** | | | | **Graphics OFF** | | | |
|---|---|---|---|---|---|---|---|---|
| | **PC 1** | | **PC 2** | | **PC 1** | | **PC 2** | |
| | ***Original Serial code*** | **Improved Serial code** | ***Original Serial code*** | **Improved Serial code** | ***Original Serial code*** | **Improved Serial code** | ***Original Serial code*** | **Improved Serial code** |
| **Wall-clock Time (s)** | *175.67* | **83.00** | *72.67* | **48.17** | *127.00* | **48.07** | *38.08* | **19.46** |
| **% change** | **- 52.75 %** | | **- 33.71 %** | | **- 62.15 %** | | **- 48.9 %** | |

As can be seen from Table 4.2, the improved serial code significantly reduces the wall-clock times of running Benchmark test case 1. Improvements of 52.75% on PC 1 and 33.71% on PC 2 were achieved when the visualisation features were applied. When the graphics were turned off, the improved serial code was better than the original serial code by 62.15% when PC 1 was used and by 48.9% on PC 2.

Similarly, Table 4.3 illustrates the Comparison between the original serial code and the improved serial code for Benchmark test case 2.

**Table 4.3 - Comparison between the original serial code and the improved serial code for Benchmark test case 2**

| | Graphics ON | | | | Graphics OFF | | | |
|---|---|---|---|---|---|---|---|---|
| | PC 1 | | PC 2 | | PC 1 | | PC | |
| | *Original Serial code* | Improved Serial code | *Original Serial code* | Improved Serial code | *Original Serial code* | Improved Serial code | *Original Serial code* | Improved Serial code |
| **Wall-clock Time (s)** | *758.33* | **338.33** | *307.50* | **166.50** | *608.67* | **192.00** | *234.00* | **97.50** |
| **% change** | **- 55.38 %** | | **- 45.85 %** | | **- 68.46 %** | | **- 58.33 %** | |

Running the improved serial code on Benchmark test case 2 still shows a marked improvement from running the same test case on the original serial code. Table 4.3 shows the timings from using the improved serial code are on average half of what were obtained from using the original serial code, whichever the processor used and whether the test cases were run with or without the visualisation features.

Table 4.4 illustrates the Comparison between the original serial code and the improved serial code for Benchmark test case 3.

**Table 4.4 - Comparison between the original serial code and the improved serial code for Benchmark test case 3**

| | Graphics ON | | | | Graphics OFF | | | |
|---|---|---|---|---|---|---|---|---|
| | PC 1 | | PC 2 | | PC 1 | | PC | |
| | *Original Serial code* | Improved Serial code | *Original Serial code* | Improved Serial code | *Original Serial code* | Improved Serial code | *Original Serial code* | Improved Serial code |
| **Wall-clock Time (s)** | *1472.5* | **1102.50** | *782.50* | **444.50** | *1254.50* | **989.50** | *719.50* | **389.00** |
| **% change** | **- 25.13 %** | | **- 43.19 %** | | **- 21.12 %** | | **- 45.93 %** | |

The results obtained from using the improved serial code on Benchmark test case 3 still show improvements from using the original serial code. PC 1 produced an improvement of 25.13% and of 21.12% when the graphics were on and off respectively. However, the same test case shows even better improvements when run on PC 2, namely 43.19% with the graphics switched on and 45.93% without the visualisation features.

Applying the serial enhancements, these results show a big improvement in the wall-clock time obtained from using the software in serial. Hence, the wall-clock time of the simulation was significantly reduced just by optimising the serial codes even before the parallelisation of the software.

#### 4.2.5.4 *Parallel movement algorithm*

The algorithm for the parallel implementation is essentially the same as the enhanced serial one apart from that the population is looped over in sub-groups related to their proximity to sub-domain boundaries. These are the high-boundary group, the low-boundary group, and the non-boundary group. The simulation clock is incremented 1/12s and the ticker is incremented by one tick. A more detailed explanation is available in Appendix 9.2.1.3. A brief description of the algorithm is illustrated in Figure 4.11:

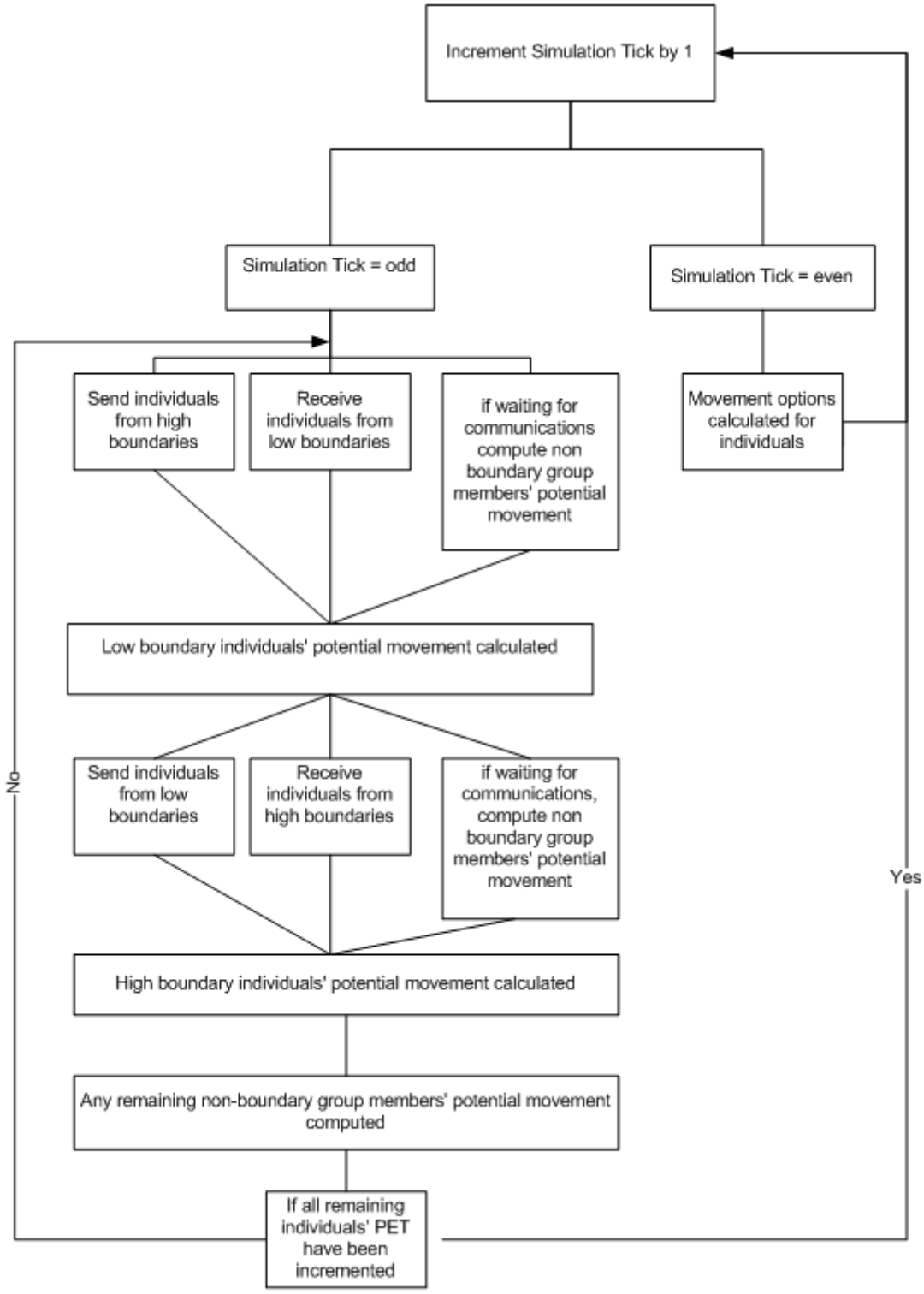



**Figure 4.11 – Parallel movement algorithm**

#### 4.2.5.5 *Synchronisation*

The communications processes mentioned in the previous sections had to be carefully orchestrated to allow the smooth transmission of the data. It is not the case that a processor can simply send and receive messages to each other. The communication needs to be arranged to ensure that the appropriate processor is ready to receive a message when the corresponding neighbouring processor is sending that message. Failure to correctly orchestrate this communication can lead to a deadlock. This occurs when both processors are waiting to receive a message from each other but they will wait indefinitely as the message will not be sent until a message is received from the other processor. More explanation about this topic is available in Appendix 9.2.1.4.

### *4.2.6 Output files*

#### 4.2.6.1 *Graphical output*

As the simulation has been distributed over a number of processors, the visualisation needs to be adapted to display the simulation on the 'Master' processor from the computations that are being calculated on the remote 'slave' processors. This is achieved by maintaining the entire nodal geometry on the 'Master' computer and updating these nodes with information from the slave processors. It is further expected that it will not be necessary to maintain the entire nodal geometry on the master computer in the future. The implementation of high end graphics is considered to be beyond the present scope of this work. All the testing has therefore been performed with the graphical output switched off. More information can be obtained from Appendix 9.2.1.5.

4.2.6.2 *Results output*

Typically, it is the 'Master' processor which handles the I/O and problem distribution. The other processors (Slave) are exactly the same as one another and only differ slightly from the master. Both the master processor and the slave processors run its portion of the simulation and it is the master process which is responsible for the collection of the results to a single output file.

### *4.2.7 Handling of the domain*

4.2.7.1 *Whole geometry on each processor*

Whenever the whole domain can be held on each processor, it is easier to load the whole geometry on each computer and then partition it. When the different partitions are found and assigned, instead of deleting the remaining nodes that are not in each processor's allocated domain(s), the algorithm keeps them for ease of the simulation. All the nodes are linked and by deleting some of the nodes, some properties might be lost and more communications and calculations are necessary. An example is the potential values that each node holds. Each node gets its potential value by its neighbouring node and this chain reaction means that deleting a section of the geometry and hence the nodes in that region, those potential values are lost unless some communication and calculations are performed to obtain those potential values. The nodes' potential values may need to be changed during the course of the simulation if ever an evacuation scenario occurs which makes an exit or several exits become closed. Whenever this situation happens, all the nodes' potential values are bound to change and need to be updated. By keeping the whole region on each processor, these extra calculations and communications can be averted and hence avoiding the complications that the absence of some nodes can incur.

It should be noted that the nodes residing outside each processor's domain are only kept as static nodes whereby they are only represented and do not handle any evacuation dynamics during the simulation unlike the dynamic residing nodes, which permit the movement of the population as well as any calculations needed related to the hazard environment. These static nodes only hold the relevant attributes that are useful to the local nodes (such as their potential values, positions

etc). Hence, they only exist as lower cost objects that passively contributes to the running of the simulation without taking as much memory requirements as a normal dynamic node requires.

All the large-scale test cases investigated so far are able to be loaded on a processor, hence not requiring the need to only load parts of the geometry on each processor. However, there is no upper limit on the area of a geometry and there might come a point when a geometry will not fit on one processor.

#### 4.2.7.2 *Parts of the geometry on each processor*

For the scalability of the problem size, an alternative to holding the whole geometry on each processor was devised. The choice to only hold parts of the domain on each processor is achievable. However as mentioned above, this alternative is not to be recommended for the various reasons provided.

##### 4.2.7.2.1 Creation of mini cloned nodes

Mini cloned nodes are objects created to represent the nodes that are present in the buildingEXODUS software. When the geometric file is loaded, the software reads the relevant data file and then creates the nodes with all the necessary attributes. By selecting only the attributes required to just create a geometric layout without the other attributes that a normal node possess (e.g toxicity values, node's colour), a set of matching cloned nodes are instead created in the software. These cloned nodes only hold the attributes necessary to partition the geometry and for that domain to function independently. The creation of a normal node in buildingEXODUS requires 152 bytes compared to 88 bytes for the creation of a minified cloned node which only holds the attributes necessary to build the geometry (such as (x, y) coordinates, arcs, etc). Once the partitioned sections are found by using the minified cloned nodes' spatial information, their matching normal nodes are loaded from the data file. Hence, the software is able to load only parts of a geometry on each processor thus allowing vast cases previously impossible, to be represented. For these simulations, whenever there is the need to get data from the nodes residing on other processors, there is no choice but to communicate the relevant information across.

However, due to time constraints more research needs to be done in this area as it has not been fully explored for any eventual problems that may arise when using this strategy. Hence, currently the whole geometry is loaded on each processor.

## 4.3 Other Parallelisation issues

### 4.3.1 Size of data transfer of each individual

In EXODUS, each occupant has a large amount of data associated with it. Each occupant has various attributes that make up a person and one major problem in the parallelisation process was the transfer of those data when the occupants needed to be communicated across whenever they crossed boundaries. Each individual has both static attributes (age, gender, etc) associated with them and dynamic attributes (waiting time, toxic gases doses, etc). In total there are 19 static and up to 50 dynamic attributes associated with each occupant. This means that communicating each individual represents a potentially significant cost as individuals cross over boundaries. In order to alleviate this communication problem, the following strategy was adopted:

- Static data is sent once when an individual initially meets a boundary.
- Dynamic data is only sent if it has changed since the last communication between the sub-domains.

Using this strategy it was also necessary to communicate whether or not a particular data item had been communicated. This was achieved by maintaining a transfer status for each data item and recording whether it had been communicated since its last change. The transfer status would then need to be part of the sent communication. This transfer status was stored as a single bit (0 or 1) which potentially reduces a transfer of 32 bits (the size of an integer or floating point number to represent an attribute) to a single bit.

### *4.3.2 Processors sharing the same exit*

In EXODUS, exits are special types of nodes and due to the nature of the evacuation movement algorithm where it is the node that initiates the movement of people, an exit can only exist on one processor and cannot be halo exits. So, when partitioning the problem, exits must not lie on the boundaries or too near to one so that they will become halo nodes or too many synchronisations and communications will occur at that location. This will inevitably be a negative impact on the goals that the parallelisation of the software are trying to achieve.

### *4.3.3 Thin partitions*

At each boundary of the partitions, there are halo nodes to enable the communications of the occupants residing on these areas. Two sets of nodes make up each halo boundary. A problem arises when the partition created has a thin region (consisting of four nodes or less) and a conflict arises with the allocation of the halo nodes. As can be seen from Figure 4.12, P1 has the same set of InnerHalo2 nodes to both P0 and P2 and this creates complications in the codes as the halo nodes have to be distinct to a particular processor. An algorithm was devised to check the width of each partition and if there are any regions which are considered thin, they are merged with the neighbouring region.

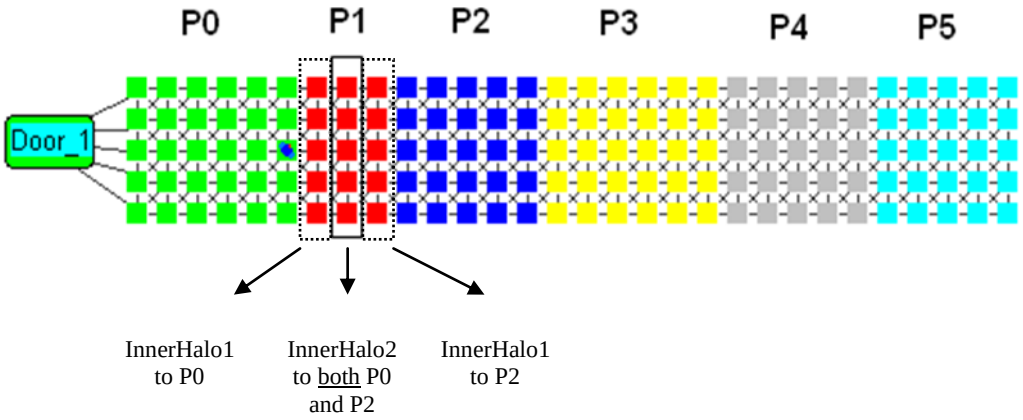



**Figure 4.12 - Diagram showing a thin partition unable to create distinct halo cells to a particular processor**

### *4.3.1 Process of Loading up of the problem in parallel*

The final process for Parallel buildingEXODUS in loading up a problem in parallel is depicted in Figure 4.13.

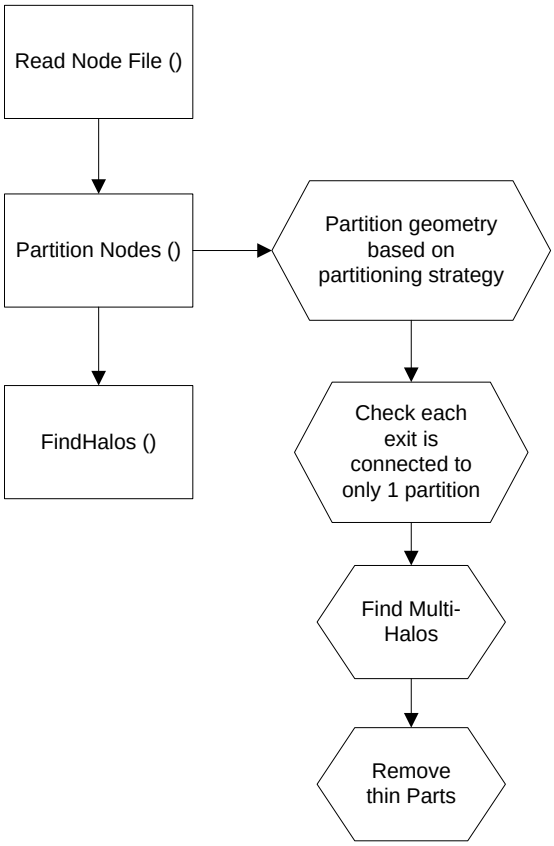



**Figure 4.13 - Loading up of the problem in parallel**

## *4.4 Approaches to assess the parallel performance of a simulation*

Quinn et al. [89] and Karavakis [90] assessed the parallel performance of their parallel system by investigating a time-step out of the whole simulation and as explained in Sections 2.6.1.1 and 2.6.1.2, these methods do not reflect an accurate parallel performance of the entire simulation.

A determining factor affecting the movement calculations of a population is the physical conditions the individuals are exposed to. Performing the movement updates of a population in a packed condition is significantly greater since there is congestion involved.

A test was devised to show the impact on the workload required to run a congested evacuation compared to a free flowing evacuation. The idea is to run two test cases to assess the impact of having congestion compared to free-flow motion on a parallel system. Similar tests were performed to how Quinn et al. [89] and Karavakis [90] measured their parallel performance.

i. The number of movement updates of the whole population is measured in a second time frame, which is how Quinn et al. [89] used as their testing criteria.

ii. Another measurement criterion was carried out to match the approach of timing a simulation second similar to what Karavakis [90] employed.

All the tests were run on a cluster of processors [1] ranging from one to five processors.

Test Case 2 illustrated in Figure 4.14 is randomly populated by 100,000 individuals to allow a 3.8 nodes per person density. This scenario can be considered as a sparse people density situation. Each person was randomly assigned a response time between 0s and 30s.

**Test Case 2 – 1,000m x 100m geometry sparsely populated by 100,000 randomly placed individuals.**

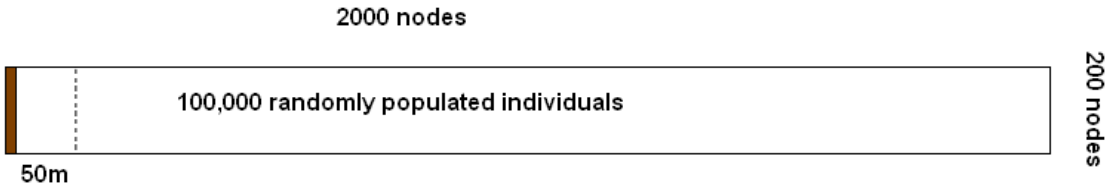



**Figure 4.14: 1,000m x 100m geometry randomly populated by 100,000 individuals**

Test Case 3 involves the same geometry involved above but compactly populated by the same population size so that there is one node per person density. This scenario can be considered a packed people density situation, as shown in Figure 4.15.

[1] *Operating System: Windows XP Professional x64 Edition (5.2, Build 3790) Service*
*Processor: Intel(R) Pentium(R) III Xeon processor (2 CPUs), ~3.2GHz*
*Memory: 8,110MB RAM*

**Test Case 3 – 1,000m x 100m geometry, populated by 100,000 individuals compactly placed in a region with a 1 node per person density**

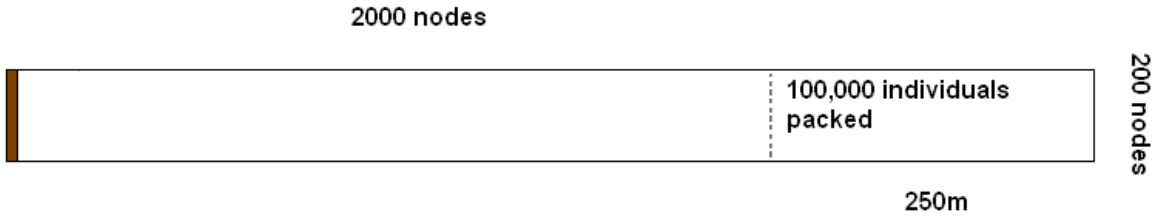



**Figure 4.15: 1,000m x 100m geometry compactly populated by 100,000 individuals**

The results were taken at a simulation time of 30s to ascertain that all the 100,000 individuals have started moving since the population has different response times ranging from 0s to 30s. Figure 4.16 demonstrates the number of movement updates needed to move the 100,000 individuals in one second when in sparse and packed conditions. Those values were taken 30s in the simulation to ensure all the occupants have started evacuating.

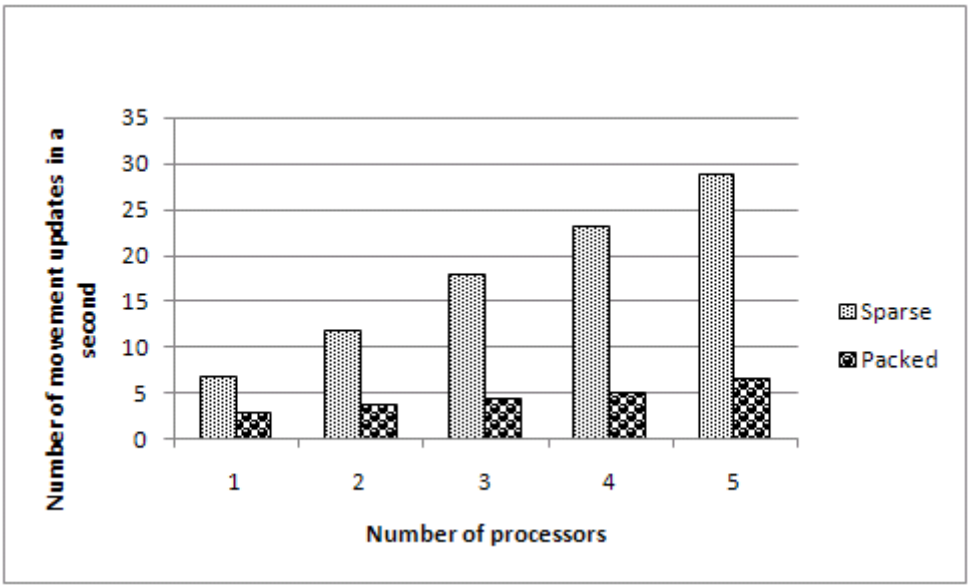



**Figure 4.16 – Comparison of the number of movement updates of 100,000 individuals per second when in sparse and packed conditions when Simulation time equals to 30s.**

Figure 4.16 clearly shows a marked difference in the number of movement updates achievable from different level of population densities. A sparse scenario generates an approximate

improvement of 5.5 times the number of processors used (5 processors generating 27.5 times more updates than a processor can) suggesting a high scalability for the parallel system. However, under packed conditions, the improvement in the number of movement of updates achievable increases roughly linearly with the number of processors used in the parallel system. These improvements still shows a trend for scalability, but not as much as the sparse scenario provides. Hence, the speedups achievable are highly dependent on the conditions of the evacuation. In this instance, a sparsely populated problem produced roughly 5 times more movement updates than a highly congested problem.

The second parallel performance criterion of timing a simulation second was also employed to the same test cases used above and the results are displayed in Figure 4.17.

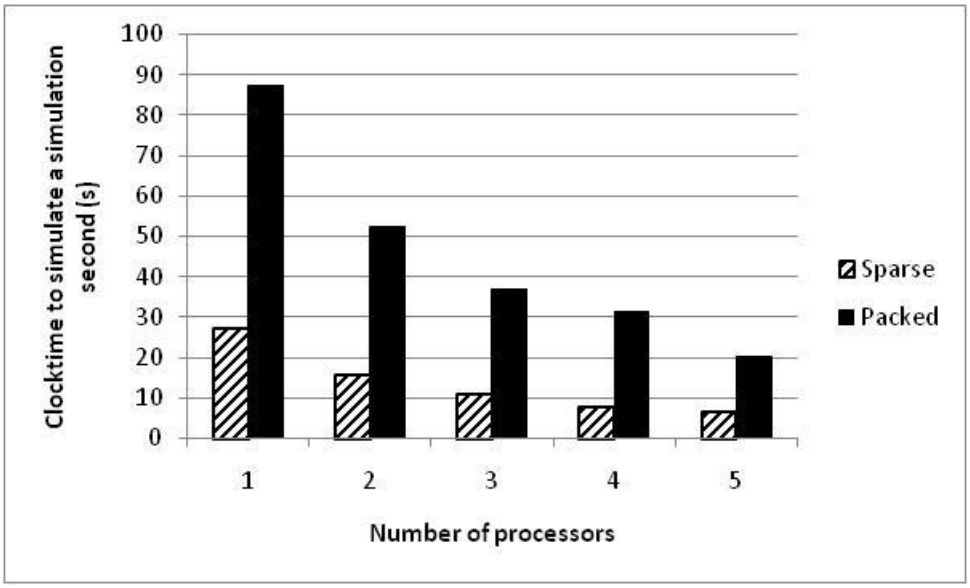



**Figure 4.17 - Comparison of the clock-time it takes to reproduce one simulation second when in sparse and packed conditions when Simulation time equals to 30s.**

For both level of congestions, a scope for scalability can be observed from Figure 4.17. As predicted, a simulation second takes longer (about 3.5 times more) to reproduce under a packed condition than when the problem was sparsely populated.

Using both the testing criteria demonstrates the variation in the results obtained when different scenarios are used. Hence, the results obtained are highly dependent on the scenario used and the

same problem size can lead to different parallel performance results depending on the constraints the problems are faced with.

Another influential factor affecting the movement calculations is the simulation of a population's movement at various stages of a simulation. At the start of a simulation, it is unlikely that the whole population starts evacuating at the same time and hence the movement calculations are less than when the whole population's movement is being calculated. Consequently, there are fewer interactions amongst the individuals at the start of a simulation than when the whole population is moving. Therefore, analysing only a stage of a simulation does not offer a true depiction of the entire simulation.

Test Case 3 was also used to determine whether there is a difference in the parallel performance at different stages in a simulation. The values were collected at 1 simulation-second and at 30 simulation-second in the simulation. The parallel performance criteria employed by Karavakis [90] of timing a simulation-second is now being tested with the sparsely packed problem and is displayed in Figure 4.18.

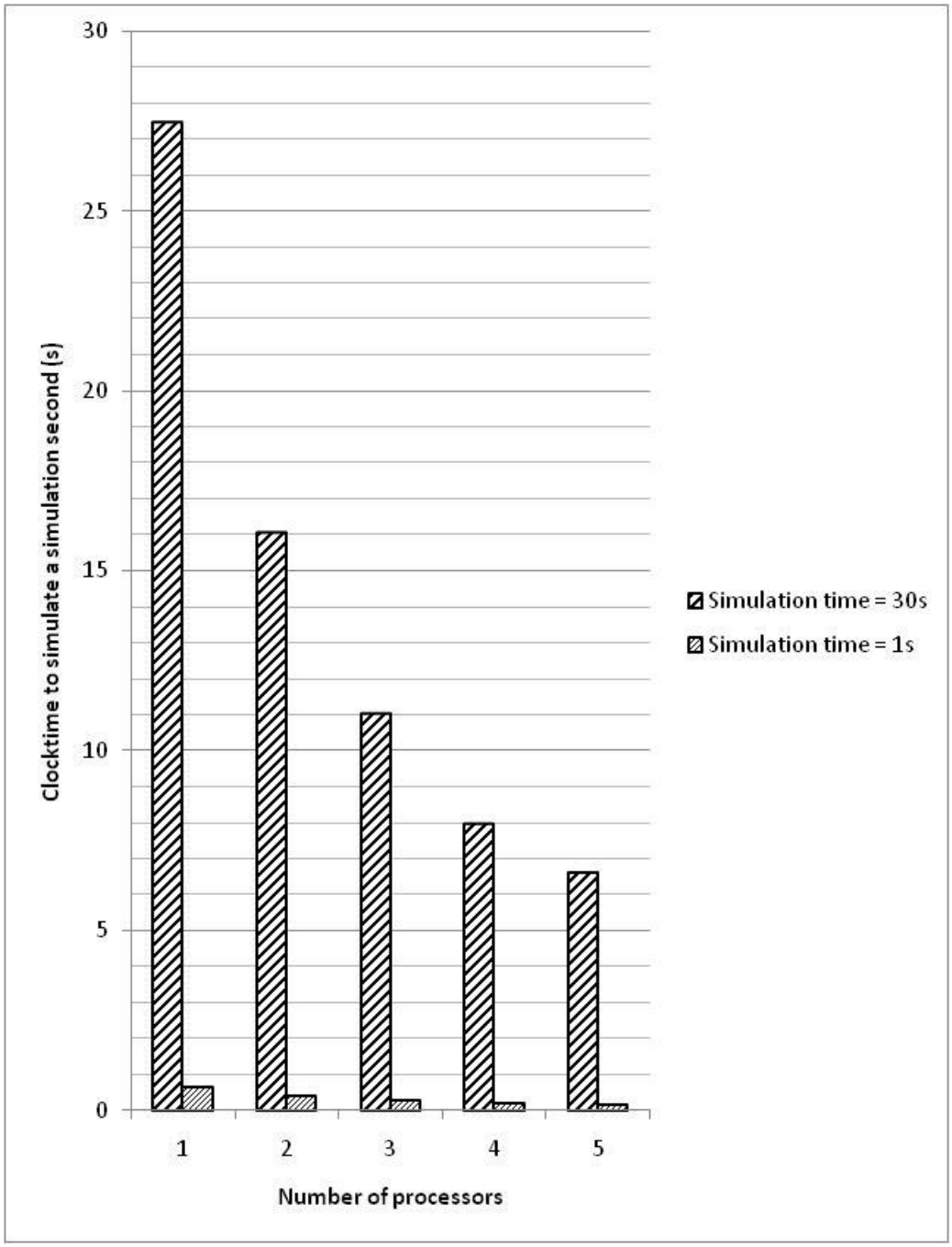



**Figure 4.18 – Comparison in the time it takes to reproduce a simulation second at different stages of a simulation (1s and 30s) in a sparse scenario**

Figure 4.18 proves there is a disparity in the time it takes to analyse a simulation second at different levels in the simulation. At the start of the simulation (1s), it takes about 40 times less time to simulate a simulation-second than when 30s in the simulation. Hence, judging a parallel performance is also dependent on when the results were taken as proven above.

Attempting Quinn et al.'s method of counting the number movement updates capable in one second, the test was run with a packed scenario and is displayed in Figure 4.19.

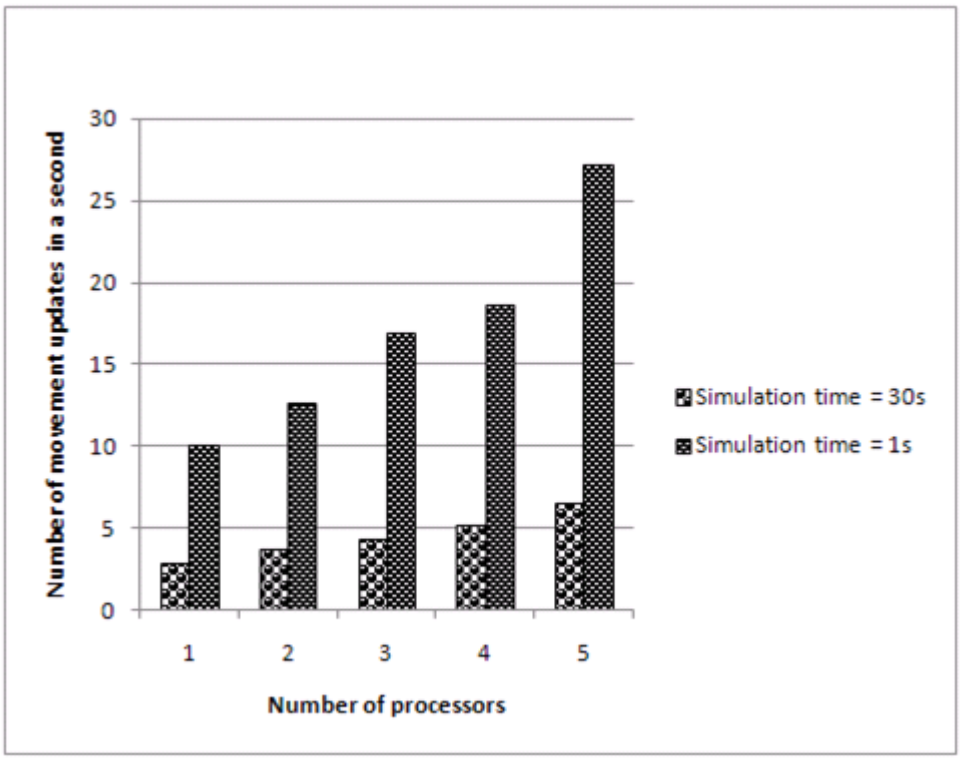



**Figure 4.19 - Comparison in the number of movement updates of 100,000 individuals at different stages of a simulation (1s and 30s) in a packed scenario.**

Here again it can be demonstrated that at different stages of a simulation, the parallel performance obtained is different. It can generate as much as 5 times more updates in a second at the start of a simulation compared to 30 simulation-second in. Hence, judging a parallel performance at a set instance of a simulation can be unreliable as the results are too varied.

A fairer and more accurate interpretation of the parallel performance is to calculate it by the total time it took to run the whole simulation rather than judging it from a time-step of the whole simulation.

## 4.5 Concluding remarks

The development and testing of miniEXODUS demonstrated that it was feasible to gain improved computational performance by creating a parallel implementation of buildingEXODUS. miniEXODUS demonstrated parallel efficiency of greater than 90% for a representative Beijing High Street geometry with 5,000 agents.

buildingEXODUS was successfully parallelised by employing the master-slave approach as parallel architecture. The processors in the cluster are connected via a LAN network and the MPI communication package enables the communications of data amongst the processors. The domain decomposition technique was chosen to partition the problem which subsequently causes a population decomposition as only the entities residing on the sub-domain belonging to a processor are computed. Halo nodes are created at the boundaries of the sub-domains to allow the communication of the individuals as they cross over those partitions. Currently, the whole geometry is loaded on each processor but a method to load up only parts of a geometry has been devised. However, due to time constraints, this technique is not fully operational as more research needs to be done into how to transmit the nodes' connectivity and interactions. The best method to assess the parallel performance is to calculate the speedup achieved by comparing the total run-time of the simulation on the parallel system to that of a processor.

# 5 DOMAIN PARTITIONING OF buildingEXODUS

The foremost element of research in this thesis is the domain partitioning of the problems loaded onto the parallelised buildingEXODUS. Various partitioning strategies need to be investigated and tested in order to come up with an optimal partitioning strategy which will maximise the speedups obtainable by minimising parallelisation costs whilst promoting load balance.

## *5.1 Partitioning Strategies*

The geometries used for the evacuation simulations in buildingEXODUS can range from regular shapes (like a rectangular domain) to irregular shapes (such as sections of a town). Another critical factor that the technique must consider is the locations of exit points. In an evacuation simulation, exit points become very busy as the simulation progresses and the partitioning strategy must take this factor into consideration when devising the partition cuts. An optimal partitioning strategy must be devised in order to cater for all possible shapes, so as to minimise the communication costs and maximise the load balance on the processors. Several partitioning strategies were devised to test those criteria.

**Please note that a cluster of processors with the following specifications were used for the parallel tests.**

***Operating System: Windows XP Professional x64 Edition (5.2, Build 3790) Service***
***Processor: Intel(R) Pentium(R) III Xeon processor (2 CPUs), ~3.2GHz***
***Memory: 8,110MB RAM***

### *5.1.1 Idealised 20 exit test case*

Test Case 4 is intended to represent an ideal case for the parallel implementation. It has been designed so that there is no boundary interaction and that the problem is well load balanced throughout the entire simulation. This test was devised to explore the upper limits of speedup potentially possible with parallel buildingEXODUS and is illustrated in Figure 5.1. A population of 100,000 individuals is uniformly distributed throughout the domain and head towards their nearest exit. Due to the geometry and exit configurations chosen there is no crossing of boundaries to reach an exit. In order to ensure the most optimal decomposition, the domain is split into 20 equal sub-domains and these are allocated to each processor. In order to ensure that the load balance is well maintained throughout the simulation, only a number of processors (1, 2, 4, 5, 10, and 20) that exactly divides into the number of domains (20) were used. The total simulated evacuation time for this scenario was 14 minutes and 26 seconds and was consistent across all the parallel simulations.

**Test Case 4 - Idealised 20 exit case**

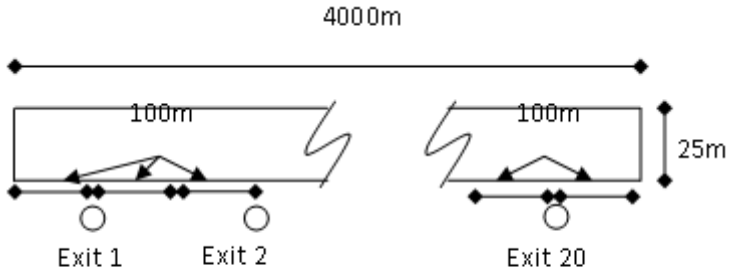



**Figure 5.1 – Geometry of Test Case 4**

5.1.1.1 *Use of single core processors:*

The results obtained for simulating the evacuation of the 100,000 occupants, when using only single core processors are displayed in Table 5.1 .

**Table 5.1 - Single core timings with speedup and efficiency values for Test Case 4**

| No. Computers | Time Taken (m:s) | Speedup | Efficiency |
|---|---|---|---|
| 1 | 58:20 | | |
| 2 | 21:40 | 2.69 | 134.50% |
| 4 | 10:00 | 5.83 | 145.75% |
| 5 | 07:55 | 7.37 | 147.40% |
| 10 | 03:49 | 15.28 | 152.80% |

These results given in Table 5.1 are impressive and are faster than expected; for example using 10 processors gives a speedup of over 15 compared to a single processor, which is known as super-linear speedup as explained in 2.3.1.

5.1.1.2 *Use of dual core processors:*

The use of a multi-core processor by Karavakis [90] produced better results than when only a single core was used. As proposed in 2.6.1.2.1, the idea of employing a cluster of dual-core processors as a parallel system was tested using Test Case 4. The above results from Table 5.1 were obtained by using only a core from the processors available whereas the results in Table 5.2 were obtained from using both cores of the same processors used above.

**Table 5.2 – Dual core timings with speedup and efficiency values for Test Case 4**

| No. Computers (dual) | Time Taken (m:s) | Speedup | Efficiency |
|---|---|---|---|
| 1 Single Core (SC) | 58:20 | | |
| 1 | 26:03 | 2.24 | 224.00% |
| 2 | 12:25 | 4.70 | 235.00% |
| 5 | 04:57 | 11.78 | 235.60% |
| 10 | 02:37 | 22.29 | 230.00% |

It can be seen from Table 5.2 that an original serial time of over 58 minutes has been reduced to below 3 minutes using 10 dual-core processors. These times are exceptionally good and are unlikely to be achieved for most practical scenarios; however, they do demonstrate a peak performance that could be attainable.

A promising finding was that using dual-core processors produced better results than using only serial core processors, which is promising as currently all the new computers include multi-core processors [90].

### *5.1.2 Equal Partitioning*

One of the partitioning strategies tested was a basic horizontal or vertical partitioning of the domain into equal sub-domains. The method involved using the geometrical 2D coordinates of the end boundaries and dividing them into the number of processors available. This method can be useful if the geometry involved is quite uniform in nature. The number of processors working on the problem can also influence a partitioning strategy.

To illustrate, consider the example in Figure 5.2:

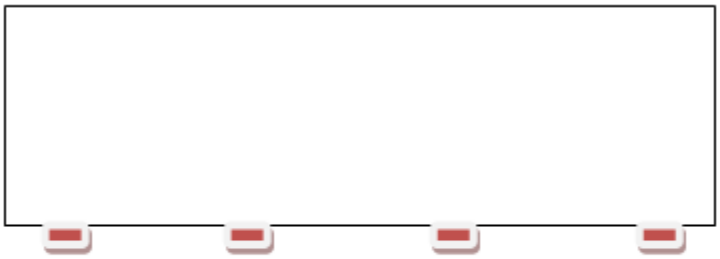


**Figure 5.2 - Geometry with four exits on one side**

For this case, the ideal partitioning strategy would be to divide it in the following manner as shown in Figure 5.3, provided there are four processors available. This case is a smaller version of Test Case 4, which confirmed this partitioning strategy was the appropriate one to use.

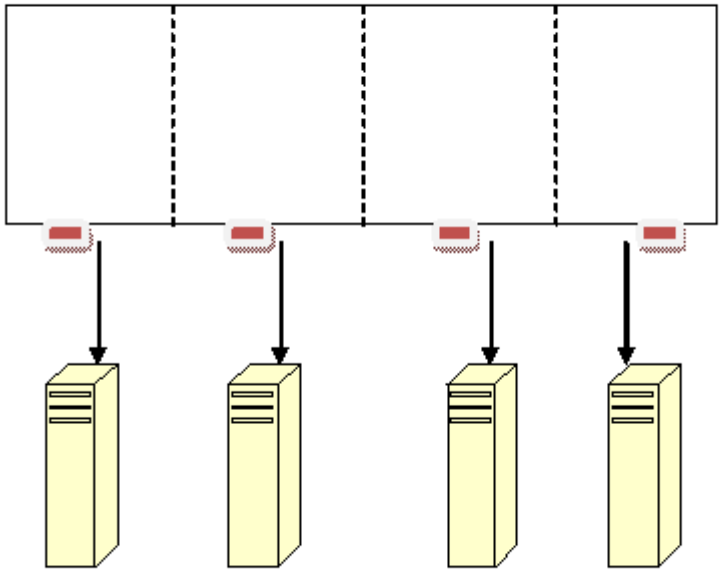


**Figure 5.3 - Geometry partitioned into four sub-domains**

This Equal Partitioning strategy works well for the above case. But a real life geometry is rarely this uniform. Most structures are complicated and the exits can be located anywhere on the domain.

Consider the following example in Figure 5.4 of an irregular domain being partitioned by the Equal Partitioning method and being simulated by four processors.

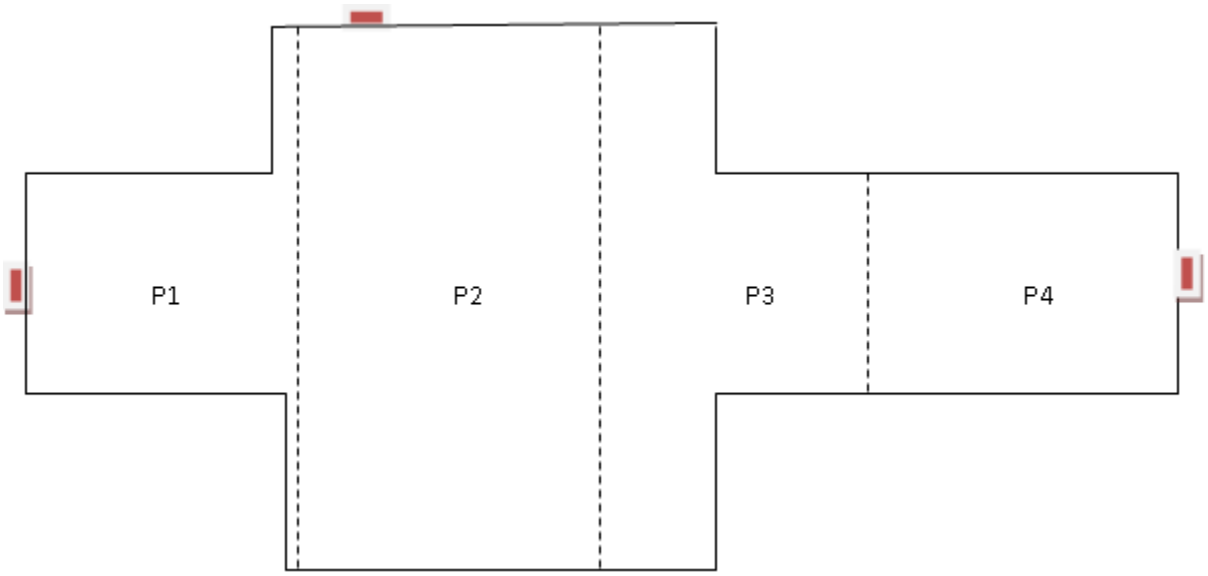



**Figure 5.4 - Irregular domain partitioned into four equal sub-domains**

Assuming the population load is evenly distributed, the above partitioning strategy will have a population load imbalance due to the different partition sizes. The sub-domain being simulated by P1 is significantly smaller than that of P2 and as the simulation progresses leading to an inefficient use of the resources available. It is expected that P1 will become inactive sooner than the other processors. So, there might not only be a load imbalance but also some processors might become idle whilst others are still working.

5.1.2.1 *Equal Partitioning tested on an irregular domain*

This partitioning technique was tested on different domains to test its efficacy.

5.1.2.1.1 Test 1 – Equal Partitioning method

**Test Case 5 - Replica of the Trafalgar square in London populated by 60,000 occupants with 14 exits in total**

Test Case 5 was tested by using two to seven processors. Figure 5.5 illustrate how the domain was partitioned into the said number of processors. The left picture is partitioned into two sub-domains, each allocated to a processor. The right hand-side picture is partitioned into seven sub-domains to be simulated by seven processors. Each processor was responsible for simulating the evacuation of individuals present in its allocated sub-domain.

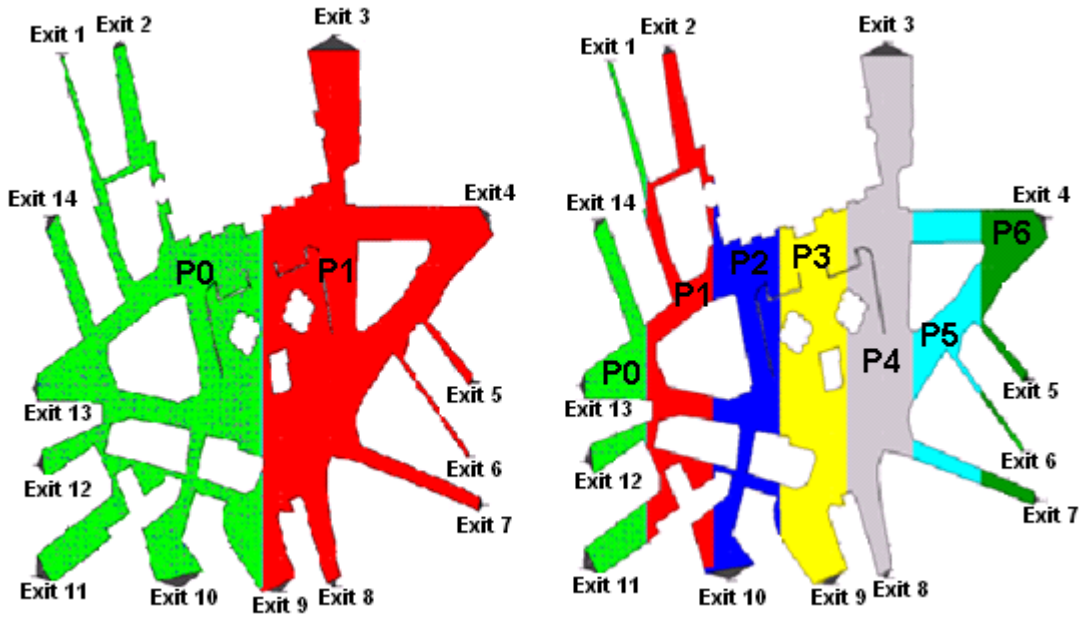



**Figure 5.5 - Equal Partitioning into two and seven sub-domains on the Trafalgar square replica**

This partitioning method does not guarantee a population load balance at the start of the simulation as this decomposition technique does not consider the layouts of the geometries. When the problem was partitioned into 2 sub-domains, it was only by chance that the resulting partitions were quite similar in sizes. However, partitioning the problem into 7 sub-domains definitely resulted in a population load imbalanced problem. The population load is illustrated in Table 5.3.

**Table 5.3 - Initial population distribution in Test Case 5**

| | **Initial population distribution** | | | |
|---|---|---|---|---|
| **Processor** | **Equal Partitioning (2 sub-domains)** | **%** | **Equal Partitioning (7 sub-domains)** | **%** |
| **0** | 28261 | 47.10 % | 5790 | 9.65 % |
| **1** | 31739 | 52.90 % | 7787 | 12.98 % |
| **2** | N/A | | 8808 | 14.68 % |
| **3** | N/A | | 12834 | 21.39 % |
| **4** | N/A | | 16308 | 27.18 % |
| **5** | N/A | | 4708 | 7.85 % |
| **6** | N/A | | 3765 | 6.28 % |
| **Total** | **60000** | **100%** | **60000** | **100%** |

From Figure 5.5 and Table 5.3, the right hand-side geometry can clearly be seen to be load imbalanced and the regions are of different sizes. For example, when 7 processors were used to partition the problem, Processor 6 only had 6.28 % of the population compared to Processor 4 which contained a population load of 27.18 %. Furthermore, some regions are not linked to an exit, meaning all its occupants will have to cross the boundaries to regions where an exit exists. This will increase the communication costs of the system as well as resulting in the processor

handling the "inside" region (coloured cyan (P5)) to become idle before its neighbouring processor handling the domain (dark green (P6)) with the exits.

The average simulation time to simulate the evacuation of all 60,000 occupants is 610s. The results are listed in Table 5.4.

**Table 5.4 - Performance values obtained by using the Equal Partitioning method on the Trafalgar square Test Case 5**

| Number of processors | Speedup | Efficiency | Parallel SimTime[1] Performance Ratio |
|---|---|---|---|
| **2** | 1.69 | 84.50% | 2.37 |
| **3** | 1.99 | 66.33% | 2.85 |
| **4** | 2.24 | 56.00% | 3.13 |
| **5** | 3.67 | 73.40% | 5.23 |
| **6** | 2.69 | 44.83% | 3.83 |
| **7** | 3.86 | 55.14% | 5.47 |

The efficiency values obtained show that the use of the cluster is not that efficient. Excepting the cases when two and five processors were used, yielding values of 84.5% and 73.4% respectively, the other efficiency values suggest that computer resources are not used efficiently incurring excessive communication and / or synchronisation costs. When six processors were used to run the simulation, an efficiency value of 44.83% was achieved. This suggests that less than half of the total processor power was being used, implying major processor idleness or high communication and synchronisation costs were involved. A possible explanation for obtaining an unscalable parallel performance can be any or a combination of the following factors; such as locations of exits on the geometry and whether each processor has roughly the same number of exits, the randomness in the population's movement that the software supports, the partitions created, the evacuation scenario being modelled (e.g, sudden closure of exits), are some of the

[1] **SimTime:** Simulation time (See Section 2.3.4)

criterion that can affect the results. On the other hand, solving numerical calculations (e.g. CFD problems) by employing parallel processing capabilities tends to give better parallel performance, since the calculations are very deterministic and uniformly distributed.

From Table 5.4, the maximum performance ratio with respect to the simulation time is 5.47, which implies that the software managed to simulate this particular evacuation 5.47 times faster that it would actually take the occupants to physically evacuate the geometry. While not sufficiently fast to provide advice to incident commanders, the performance enhancements achieved make it viable to investigate these type of evacuation scenarios for planning and possibly interactive training purposes.

The computational speedup values are illustrated on the graph shown in Figure 5.6:

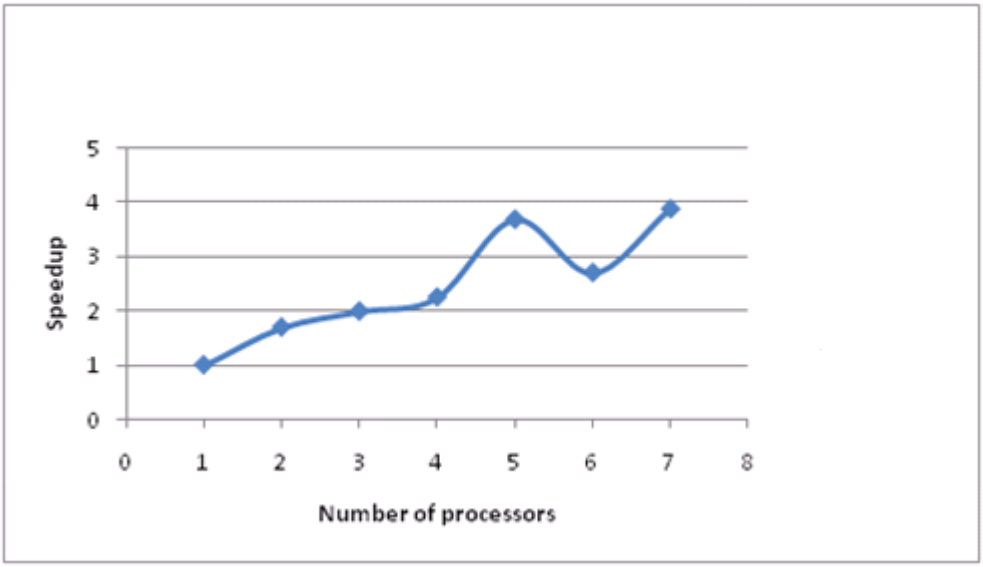



**Figure 5.6 - Speedup values from using a replica of the Trafalgar Square**

From these values, though there is a gradual increase in trend in the speedup values, there does not seem to be a simple positive relationship between the speedup and the number of processors. In this case, using six processors gave worse results than using five, which is not implying an increasing trend in the speedup values. This does not guarantee that increasing the number of processors will incur an increase in speedup and therefore produce a shorter run time. A perfect speedup value would be equal to the number of processors used. But due to communication and

latency costs, they are expected to be just below the number of processors used. However, these values are too low compared to the number of processors used, i.e. seven processors were used and only a speedup of around 4 was obtained. The most likely cause for these average speedup values is the load imbalance factor. It can be seen from Figure 5.5 that this partitioning strategy does not favour the load balancing of the domain. In some cases the surface area of the sub-domains are dissimilar in size as well as some sub-domains not owning any exits, hence making them empty quicker than those which have exits.

#### 5.1.2.2 *Equal Partitioning tested on a domain with only one exit*

The location of exits plays an important role in the formulation of the partitioning strategy. At exit points, it can be assumed that this area will remain active until the end of the simulation. Hence, the ideal strategy is to make each processor handle at least an exit point. However, there may be cases where the number of exits is less than the number of processors available and since an exit can only be on one processor as explained in Section 4.3.2, there is no choice but to make only one processor handle this exit. To illustrate, consider Test Case 6:

##### 5.1.2.2.1 Test 2 – Equal Partitioning method

Test Case 6 illustrates one processor handling the only exit which can negatively affect the speedups achievable if the Equal Partitioning method is used.

**Test Case 6 - 500m x 25m with one exit at one end and populated by 25,000 identical occupants with a response time of zero.**

A rectangular geometry measuring 500m by 25m was considered with a free flow exit along the whole width of the geometry. This geometry was randomly populated with 25,000 identical occupants, i.e. they all have the same attributes.

This problem is expected to be load imbalanced as well as sustaining processor(s) idleness as the simulation progresses. As the occupants move towards the exit catered for by Processor 0, the areas belonging to the other processor(s) are being evacuated. Gradually, the other processor(s)

will have fewer people than Processor 0, hence creating a load imbalance and as all the occupants have vacated the local domains, these processors will become idle as Processor 0 keeps working.

This test was conducted by using two, three and four processors as illustrated by Figure 5.7 - Figure 5.9.

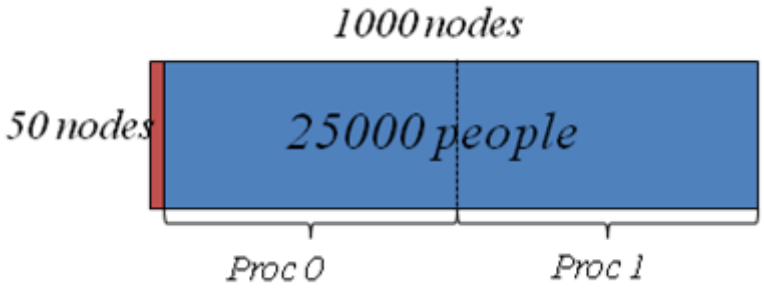



**Figure 5.7 - Equal Partitioning with using 2 processors**

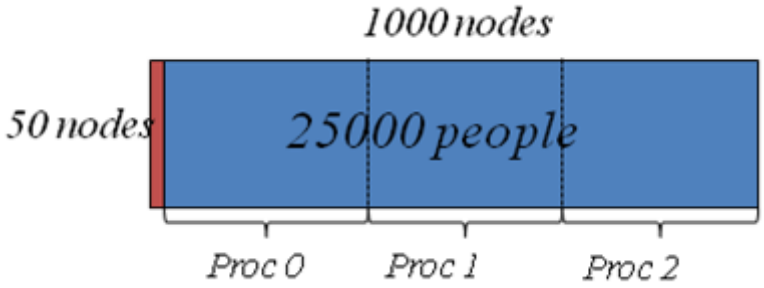



**Figure 5.8 - Equal Partitioning with using 3 processors**

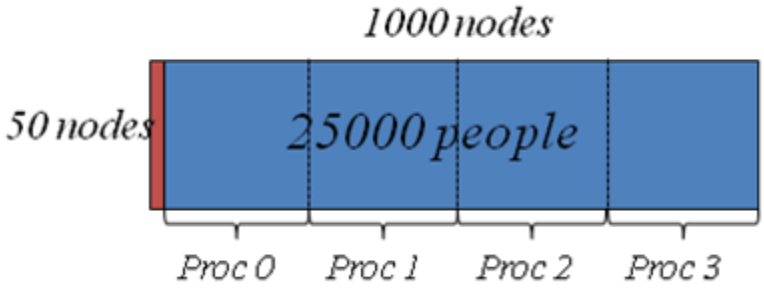



**Figure 5.9 - Equal Partitioning from using 4 processors**

The average simulation time to evacuate the population of 25,000 occupants is 833.5s. The speedup and efficiency values achieved by running those tests are shown in Table 5.5:

**Table 5.5 - Equal Partitioning method - speedup and efficiency values**

| Number of processors | Partitioning ratio | Theoretical Speedup from Eq (4) | Speedup | Efficiency |
|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 100% |
| 2 | 0.5 : 0.5 | 1.33 | 1.352 | 67.60% |
| 3 | 0.33 : 0.33: 0.33 | 1.80 | 1.760 | 58.67% |
| 4 | 0.25 : 0.25: 0.25: 0.25 | 2.29 | 2.10 | 52.50% |

From these results, the speedups obtained are not close enough to the ideal value of the number of processors used. The reason here is mostly due to processor idleness. For the case with four processors being used Processor 3 becomes idle shortly after the simulation starts and then Processor 2 follows soon after, and so on. So, the optimum usage of the available processors is not being exploited.

#### 5.1.2.3 *Equation to find the partition cut*

Using the Equal Partitioning technique (Section 5.1.2) for cases where there is only one exit eventually results in a load imbalance and hence a mediocre speedup. A possible solution to that problem is to assign bigger parts to the processors the farther their domain is from the exit. This tactic should prolong the time it will take the end processors to become idle and hence the load imbalance will happen at a later stage than it would normally have been if the Equal Partitioning method had been used. An illustration of this approach is presented in Figure 5.10. This illustrates when N processors are used to partition the geometry into N parts with different sizes with ratios $F_1$ to $F_N$.

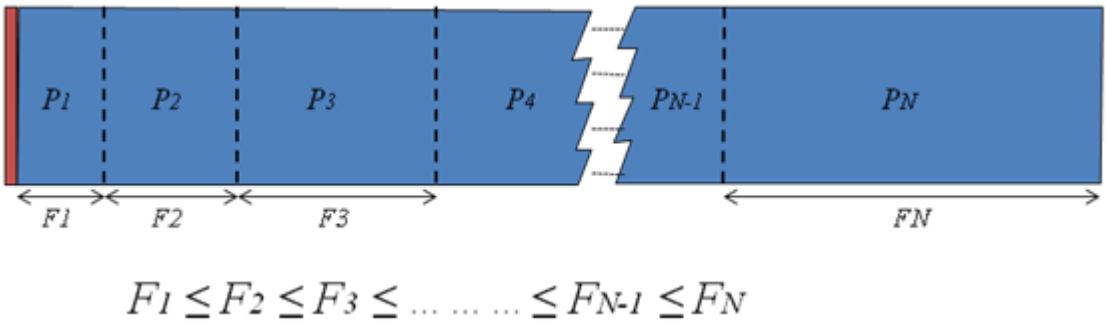


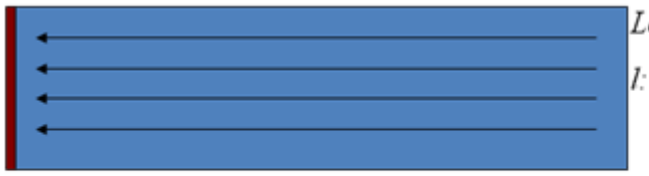

**Figure 5.10 - Partition cut ratio**

5.1.2.3.1 Derivation of a formula to work out the partition cut ratios

Using the same principles derived by Dr. Grandison (Section 9.3) used to work out the estimation of the workload and thus the speedup, a formula was derived to estimate the workload of the system when different sized domains are used.

#### 5.1.2.3.1.1 Partition cut formula excluding costs

The following formula was derived by ignoring any costs involved in the parallel system. A rectangular geometry was considered with a free flow exit along the whole width of the geometry, as shown in Figure 5.11. The people's movement was assumed to resemble fluid flow.




$$Workload_{serial} = Average\ population\ X\ Average\ distance\ travelled$$
$$= \frac{Z(1)+0}{2} \times \frac{(l)}{2} = \frac{Zl}{4}$$

**Figure 5.11 - Geometry used to derive the partition cut equation**

The full derivation for Eq (4) can be found in Section 9.3 from the Appendix.

$$\text{Total workload} = \frac{Zl}{4}\left(2\sum_{i=1}^{i=N-1} F_i^2 + F_N^2\right) \qquad \textbf{Eq (4)}$$

Minimising Eq (4) provides the optimal partition sizes that should maximise the speedup as shown in Section 9.3. Hence in order to optimise the problem, the ratios of the parts should be in this decomposition: [1 : 1 : 1..... : 2]

This optimal decomposition implies that all the partitions should be of the same sizes except for the end partition being twice their sizes. Hence the assumption that each part will increase in size the farther they are from the exit is not true according to Eq (4).

This formulation was tested from 'Test Case 6 - 500m x 25m with one exit at one end and populated by 25,000 identical occupants' used to test the Equal Partitioning method. The average simulation time to evacuate the population of 25,000 occupants is 833.5s. The speedup achieved is shown in Table 5.6:

**Table 5.6 - Partition cut (without communication costs) partitioning method - speedup and efficiency values**

| Number of processors | Partitioning ratio | Theoretical Speedup from Eq (4) | Speedup | Efficiency |
|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 100.00% |
| 2 | 0.33 : 0.67 | 1.50 | 1.527 | 76.35% |
| 3 | 0.25 : 0.25 : 0.5 | 2.00 | 1.935 | 64.50% |
| 4 | 0.2 : 0.2 : 0.2 : 0.4 | 2.50 | 2.28 | 57.00% |

A slight improvement can be observed in comparison with those obtained when using the Equal Partitioning method. The comparison is given in Figure 5.12:

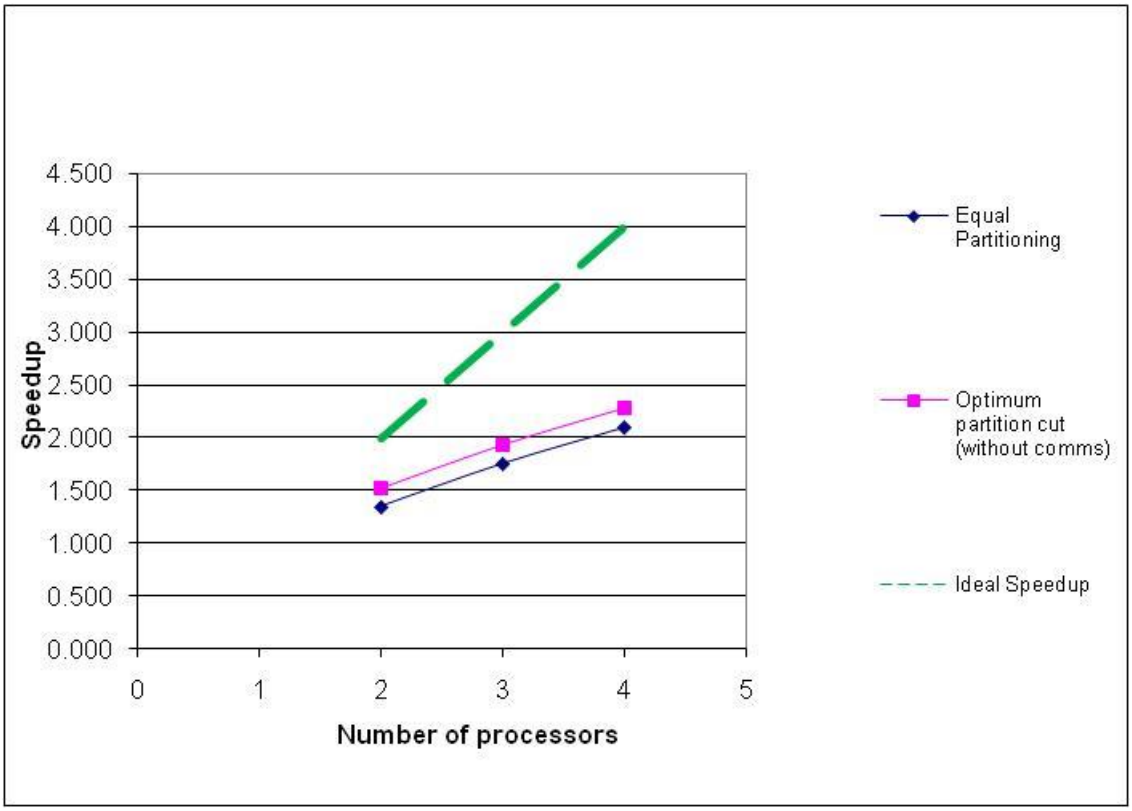



**Figure 5.12 - Comparison in the speedup values obtained from using different partitioning technique**

This slight improvement in the speedup values obtained from using the optimal decomposition ratio obtained from minimising Eq (4) suggests that the equation is usable. The actual speedup values are just below those predicted by the equation. This is expected as Eq (4) does not take into consideration the parallelisation costs.

#### 5.1.2.3.1.2 Partition cut equation including costs

Consequently, an attempt at improving Eq (4) so that it incorporates the parallelisation costs is devised. The same methodology used to find Eq (4) is used to incorporate the costs. The equation now becomes:

$$\text{Total workload} = \frac{Zl}{4}\left\{F_N^2 + 2\sum_{i=1}^{N-1} F_i^2 + 4k\left[F_N + 2\sum_{i=2}^{N-1} F_i\right]\right\} \qquad \text{Eq (5)}$$

A detailed derivation to Eq (5) is given in Section 9.4 in the Appendix. This equation can be minimised to find out the ratios of the sub-domains, hence maximising the speedup. The ratios obtained now suggest that the partition nearest the exit is slightly larger than the adjacent partitions, with the end partition being nearly twice the size of the first partition. The same 'Test Case 6 - 500m x 25m with one exit at one end and populated by 25,000 identical occupants' that were used in the previous section were used to test this new equation. The average simulation time to evacuate the population of 25,000 occupants is 833.5s. The speedup values obtained shown in Table 5.7:

**Table 5.7 - Optimal partition cut (with communication costs) partitioning method –- speedup and efficiency values**

| Number of processors | Partitioning ratio | Theoretical Speedup from Eq (5) | Speedup | Efficiency |
|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 100.00% |
| 2 | 0.35: 0.65 | 1.380 | 1.513 | 75.65% |
| 3 | 0.26 : 0.24 : 0.5 | 1.814 | 1.899 | 63.30% |
| 4 | 0.22 : 0.19 : 0.19 : 0.4 | 2.234 | 2.181 | 54.53% |

Figure 5.13 gives a comparison of the speedup values obtained from using the different partitioning methods attempted so far. It also shows the ideal speedup that the system should be producing.

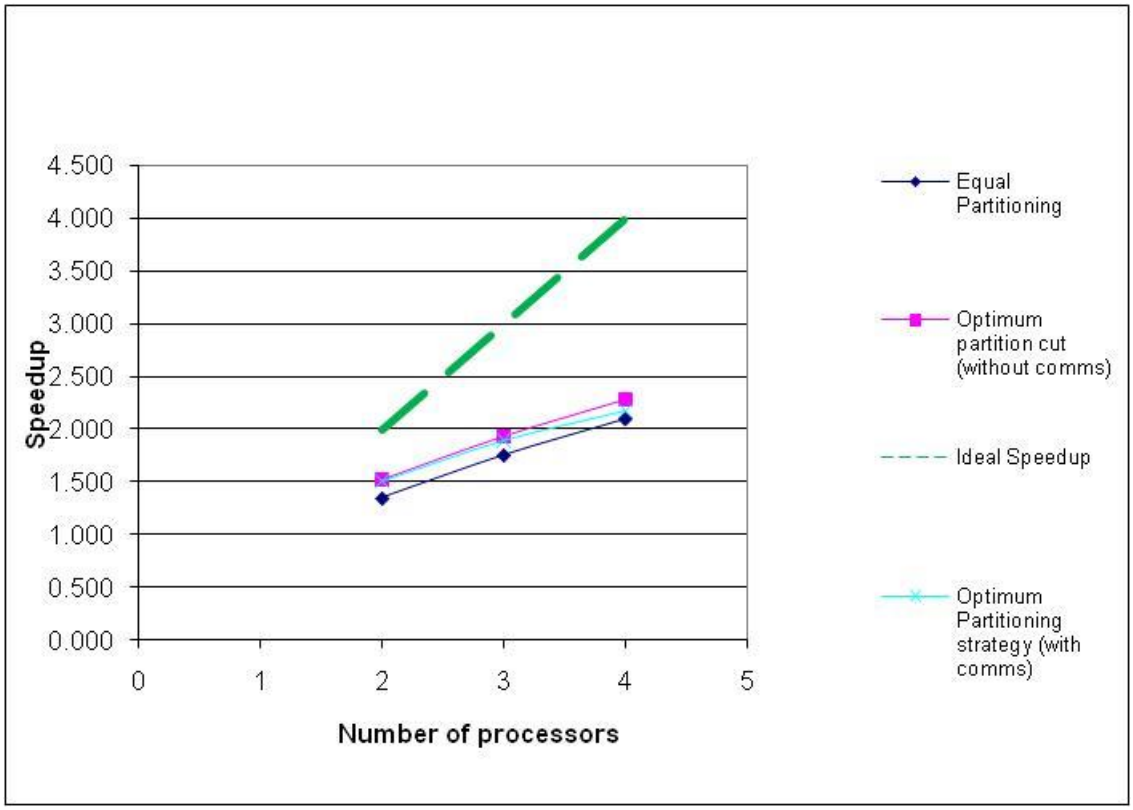



**Figure 5.13 - Comparison in the speedup values obtained from using different partitioning techniques**

The new proposed partitioning ratio did not yield any drastic improvements and the speedup values obtained are quite similar to the ones obtained from using Eq (4). Hence, Eq (5) was abandoned in the search for having an equation to predict an optimal partitioning ratio to partitioning a geometry. Comparing with the previous two methods examined, Eq (4) seems not to have been significantly bettered. Hence the 'Equal Partitioning technique' can be potentially modified to improve the results and is explained in 5.1.3.

### *5.1.3 Cyclic Equal Partitioning*

From the previous assumptions and tests, the problem is believed to be related more to load imbalance rather than communications costs. The previous attempts to find the optimal number of partitions to minimise the communication costs was by having the number of partitions equal to the number of processors. The load imbalance and processor idleness suffered, because as soon as the population evacuated the domain of one of the slaves, this processor gradually became load unbalanced and eventually became idle whilst the other processors continued working. A possible solution to solve the load imbalance and processor idleness problem is to create more regular regions and intelligently allocating them to processors, hence tackling the load imbalance problem as well as delaying processor idleness.

#### 5.1.3.1 *Equation to estimate the speedup achievable*

This Section (5.1.3.1) outlines the equation which was developed by Dr Grandison, and this work is used in the later stages of this thesis, hence the reason of it being included here.

Consider a simple example involving a rectangular geometry in which there is an exit at each end of the geometry and the population is uniformly distributed throughout the environment (see Figure 5.14). The population is further assumed to be smoothly flowing with no congestion. In this case if each individual heads towards their nearest exit the population will split in half, with half going to the exit on the left and half going to the exit on the right. In this case the best partition for two computers would be achieved by simply splitting the domain in half. (Note: This simple problem could in fact be separated into two completely separate problems without the need for a parallel implementation.)

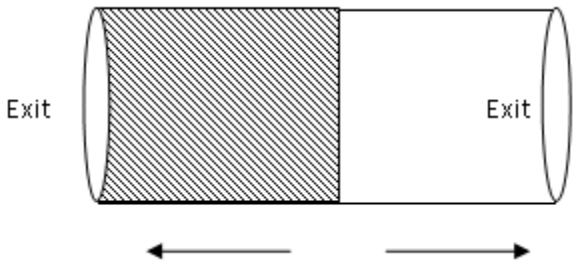



**Figure 5.14 – Evacuation domain consisting of rectangular geometry, two exits and uniform population distribution split into two sub-domains**

However, if one of the exits were removed, perhaps due to some emergency event, the simple partitioning would be inappropriate as one of the computers would have progressively less work to do as the population progressively moves toward the only exit (see Figure 5.15). A point would be reached when the processor handling the right sub-domain would be idle while the processor handling the left sub-domain would still be working hard.

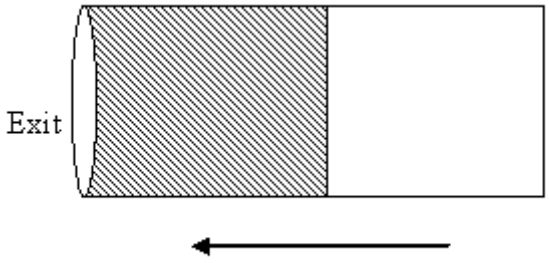



**Figure 5.15 – Evacuation domain consisting of rectangular geometry, one exit and uniform population distribution split into two sub-domains**

Another problem with this simple partition is that it only applies to two processors and cannot be generalised for an arbitrary number of processors. If the population was not initially uniformly distributed throughout the domain this would also lead to a computational load imbalance.

These problems can be mitigated by using multiple sub-domains per computer as illustrated in Figure 5.16. In this decomposition the shaded areas are computed on one processor and the non-shaded areas are computed on another. Using this scheme the workload is more evenly balanced.

As the population moves toward the exit both processors are generally kept busy, it is only when the geometry has emptied to the last sub-domain that the second processor becomes idle.

Figure 5.16 shows the domain being split into 6 sub-domains.

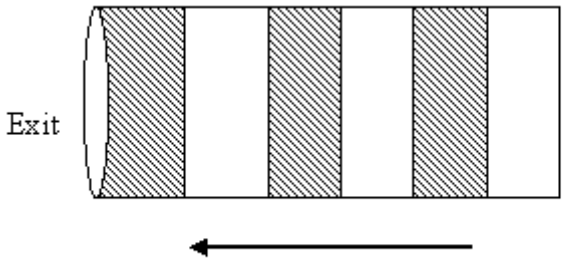



**Figure 5.16 - Evacuation domain consisting of rectangular geometry, one exit and uniform population distribution split into multiple sub-domains**

This methodology can be extended to using any number of processors and on any arbitrary geometry using any decomposition strategy as illustrated in Figure 5.17. For the above partition for two processors the partition can be represented as 121212, if three processors were being used then the partition would be represented as 123123. With this technique the computational load on each computer is kept well balanced and the load is more evenly balanced with more partitions per processor. Eq (6) was derived to consider any arbitrary number of processors and sub-domains, the full derivation is given in Appendix 9.5.

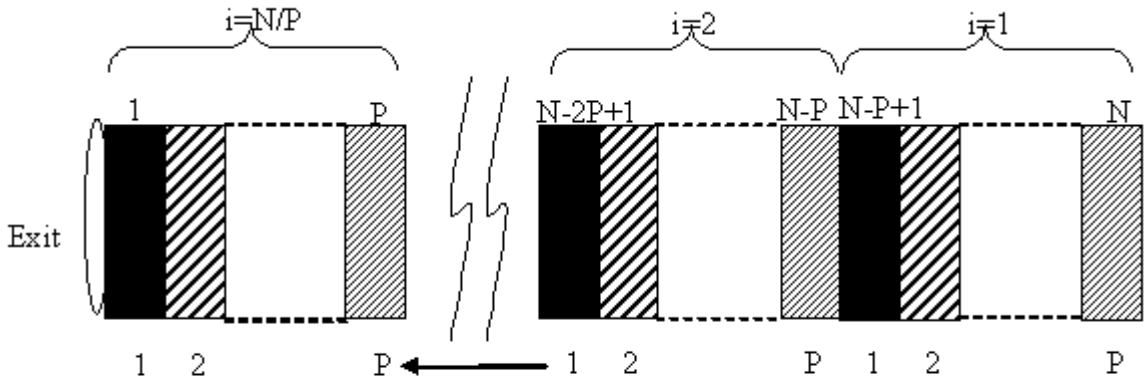



**Figure 5.17 - Generalised domain decomposition for N sub-domains and P processors**

The estimated speedup Eq (6) derived from Dr. Grandison does not include any costs (such as communication costs) associated with a parallel system. Consequently, this equation was used as a basis in an attempt to derive a more accurate equation to estimate the speedup as demonstrated in the following Section 5.1.3.2.

$$Estimated\ Speedup = \frac{NP}{N + P - \mathbf{1}} \qquad \textbf{Eq (6)}$$

5.1.3.2 *Equation including costs to estimate the speedup achievable*

An attempt to account for communication costs was devised using equation Eq (6). The formulation of that attempt can be seen in Appendix 9.6.

Let k be the communication costs associated with one person. An estimate of the speedup achievable can be approximated by Eq (7):

Legend:

*S: Speedup*

*l: Length of the domain*

*N: Number of sub-domains*

*P: Number of processors*

*k: communication costs associated with one person.*

$$S = \frac{lNP}{l(N+P-1)+4Nk(N-1)} \qquad \textbf{Eq (7)}$$

Differentiating the speedup with respect to N will give us the optimal number of sub-domains (N) to yield the maximum speedup:

$$S = \frac{lNP}{l(N+P-1)+4Nk(N-1)} = \frac{lNP}{lN+lP-l+4kN^2-4kN}$$

$$\frac{\partial S}{\partial N} = \frac{(lN+lP-l+4kN^2-4kN)lP - lNP(l+8kN-4k)}{[l(N+P-1)+4Nk(N-1)]^2}$$

$$\frac{\partial S}{\partial N} = \frac{l^2NP+l^2P^2-l^2P+4klPN^2-4klNP-l^2NP-8klPN^2+4klNP}{[l(N+P-1)+4Nk(N-1)]^2}$$

$$\frac{\partial S}{\partial N} = \frac{l^2(P^2-P)-4klPN^2}{[l(N+P-1)+4Nk(N-1)]^2}$$

At maximum S, $\frac{\partial S}{\partial N} = 0$

$$\frac{\partial S}{\partial N} = \frac{l^2(P^2 - P) - 4klPN^2}{[l(N + P - 1) + 4Nk(N - 1)]^2} = 0$$

$$l^2(P^2 - P) - 4klPN^2 = 0$$

$$4klPN^2 = l^2(P^2 - P)$$

$$4kN^2 = l(P - 1)$$

$$N^2 = \frac{l(P - 1)}{4k}$$

$$N = \sqrt{\frac{l(P - 1)}{4k}}$$

Using Eq (7), several tests were run in an attempt to estimate k. These tests were done by the following Test Case 7.

5.1.3.3 *Results from testing the Cyclic Equal Partitioning method*

5.1.3.3.1 Test 1 – Cyclic Equal Partitioning method

**Test Case 7 - Open rectangular geometry measuring 100m x 1,000m populated by 100,000 people**

This scenario consists of 100,000 people in an open rectangular geometry measuring 100 m by 1,000 m producing an area of 100,000 $m^2$ resulting in a crowd density of 1 person/$m^2$. The crowd

moves to the left to exit the geometry. The left side of the geometry is completely open, creating a 100 m wide exit (see Figure 5.18). This case can be considered as the realisation of the theoretical studies from Eq (6) and Eq (7).

The simulation time to evacuate 100,000 occupants from this geometry is 14m20s. This scenario was tested with three different types of partitioning cuts: 20, 50 and 100 equally sized sub-domains and the results are displayed in Table 5.8.

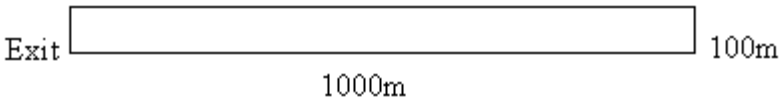



**Figure 5.18 – Large open geometry**

The speedup values obtained were used to work out an approximation of the communication costs k.

**Table 5.8 - Estimation of the communication costs from speedup values obtained**

| | Speedup | | | Theoretical Speedup From Eq (6) | | | k calculated from $\frac{1NP}{[1(N+P+1)+4Nk(N-1)]}$ From Eq (7) | | |
|---|---|---|---|---|---|---|---|---|---|
| **No. sub-domains →** / **No. Computers ↓** | **20** | **50** | **100** | **20** | **50** | **100** | **20** | **50** | **100** |
| **2** | 2.04 | 2.00 | 1.83 | 1.90 | 1.96 | 1.98 | -0.92 | -0.10 | 0.21 |
| **3** | 2.97 | 2.94 | 2.74 | 2.73 | 2.88 | 2.94 | -1.18 | -0.10 | 0.19 |
| **4** | 3.86 | 3.96 | 3.71 | 3.48 | 3.77 | 3.88 | -1.50 | -0.25 | 0.12 |
| **5** | 4.67 | 4.82 | 4.60 | 4.17 | 4.63 | 4.81 | -1.70 | -0.22 | 0.12 |
| **6** | 5.18 | 5.74 | 5.45 | 4.80 | 5.45 | 5.71 | -1.21 | -0.28 | 0.13 |
| **7** | 5.85 | 6.66 | 6.20 | 5.38 | 6.25 | 6.60 | -1.36 | -0.35 | 0.17 |
| **8** | 6.37 | 7.52 | 7.33 | 5.93 | 7.02 | 7.48 | -1.24 | -0.39 | 0.05 |
| **9** | 7.15 | 8.29 | 8.16 | 6.43 | 7.76 | 8.33 | -1.86 | -0.38 | 0.06 |
| **10** | 7.44 | 8.74 | 9.34 | 6.90 | 8.47 | 9.17 | -1.39 | -0.18 | -0.05 |

The attempt to come up with an estimate of the communication costs associated with one person did not yield a definite value for k. In most cases, k was found to be negative and hence cannot be considered a cost. Furthermore, k was found to vary as the number of sub-domains or/and the number of processors used were varied. If k was positive and quite distinctive irrespective of the number of sub-domains or number of processors used, Eq (7) would have been adopted to estimate the speedup achievable as well as finding the optimal number of sub-domains to partition the geometry.

Figure 5.19 illustrates the results obtained from the above test case.

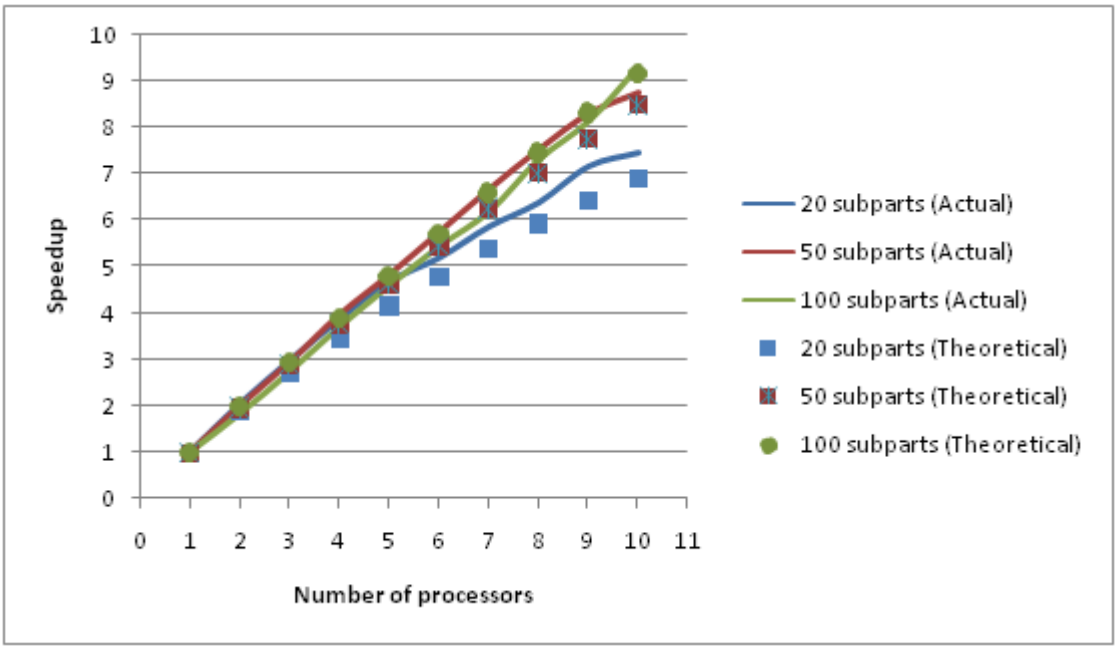



**Figure 5.19 - Comparison in the speedup values obtained from using different partitioning technique**

Theoretically 100 sub-domains should give the best speedup performance due to the best load balance being maintained throughout the simulation. In addition to maintaining the load balance, there are other factors that influence the performance including the cost of communication, which increases with additional sub-domains, and hardware effects such as increased cache size. Consequently the increased cache size made available by increasing the number of computers improves the performance beyond the theoretical prediction. The actual speedup performance is therefore a combination of the load balance and these other factors. As can be seen from Table 5.8, most the best speedup values were obtained when the geometry was partitioned into 50 sub-domains, which is in between the 20 sub-domains (with less communications and less load balanced) and the 100 sub-domains (with increased communications as well as increased load balance). It can be seen from Table 5.8 that when 2 processors are used, the best partition is 20 sub-domains. A possible explanation to this scenario is that the communication cost is far lower with 20 sub-domains compared to the partitions with a higher number of sub-domains. When 10 computers are used, the increased communication costs brought upon by the 100 sub-domains case still did not outweigh the advantages of an improved load balance, giving a speedup value of 9.34 compared to 7.44 obtained with a 20 sub-domains problem.

5.1.3.3.2 Test 2 – Cyclic Equal Partitioning method

Since no optimal number of sub-domains could be found from equation Eq (7), Similar to what was tested in 5.1.3.3.1, 'Test Case 6 - 500m x 25m with one exit at one end and populated by 25,000 identical occupants' was partitioned into different number of equal sub-domains to test the Cyclic Equal Partitioning method. The simulation time to predict the evacuation of the 25,000 occupants is 833.5s. The results are shown in Table 5.9:

**Table 5.9 - Results from using the Equal Cyclic Partitioning method on Test Case 6**

| | **Speedup** | | | **Theoretical Speedup from Eq (6)** | | |
|---|---|---|---|---|---|---|
| **No. Processors →** | **2** | **3** | **4** | **2** | **3** | **4** |
| **No. sub-domains ↓** | | | | | | |
| **5** | 1.91 | 2.48 | 2.95 | 1.67 | 2.14 | 2.50 |
| **10** | 2.10 | 2.90 | 3.61 | 1.82 | 2.50 | 3.08 |
| **15** | 2.14 | 3.06 | 3.9 | 1.88 | 2.65 | 3.33 |
| **25** | 2.16 | 3.14 | 4.10 | 1.92 | 2.78 | 3.57 |
| **50** | 2.11 | 3.07 | 4.10 | 1.96 | 2.88 | 3.77 |
| **100** | 1.92 | 2.87 | 3.85 | 1.98 | 2.94 | 3.88 |

From the results obtained, irrespective of the number of processors used, super-linear speedup values were achieved, meaning they are greater than the number of processors used. Clearly this method is the most promising of all the previous methods tried. It would seem to suggest that increasing the number of sub-domains, and hence making the system more load balanced and also delaying processor idleness at the expense of increasing the communication costs, resulted in very good speedups. The conclusion that was found was that load imbalance and processor idleness are the crucial factors which influence this problem, rather than communication costs. This is demonstrated by the fact that using 100 sub-domains, hence incurring high communication costs

whilst making the system load balanced, produced better speedup values than when only five sub-domains were used. The geometry partitioned into only five sub-domains minimises the communication costs but does not guarantee a load balanced problem. This can arise if there are sub-domains where the population exits quicker, thus making the processors handling those areas become idle.

Most of the actual speedup values obtained surpassed those predicted by the theoretical speedup Eq (6). A possible explanation is that the reduced data in each processor's cache did speed up the results. This advantage is even more appreciated, considering better speedup values achieved by having increased cache sizes were obtained, even though parallelisation costs were not included to calculate the theoretical speedup values.

Another performance analysis is the ratio of processor usage defined by the Efficiency factor. The efficiency values are displayed in Table 5.10:

**Table 5.10 - Efficiency values for Test Case 6 using the Equal Cyclic Partitioning method**

| Number of sub-domains | Efficiency | | |
|---|---|---|---|
| | 2 processors | 3 processors | 4 processors |
| **5** | 95.50% | 82.67% | 73.75% |
| **10** | 105.00% | 96.67% | 90.25% |
| **15** | 107.00% | 102.00% | 97.50% |
| **25** | 108.00% | 104.67% | 102.50% |
| **50** | 105.50% | 102.33% | 102.50% |
| **100** | 96.00% | 95.67% | 96.25% |

By having efficiency values above a 100% shows that the parallel system is functioning above capacity. By merging the processor power, the results obtained are beyond what was originally expected and this shows that the parallelisation and partitioning technique of this test case is successful. These values suggest that all the processors were constantly being used and communication and synchronisation costs were minimal compared to the total gain obtained.

A comparison to the previous tests done is illustrated in Figure 5.20.

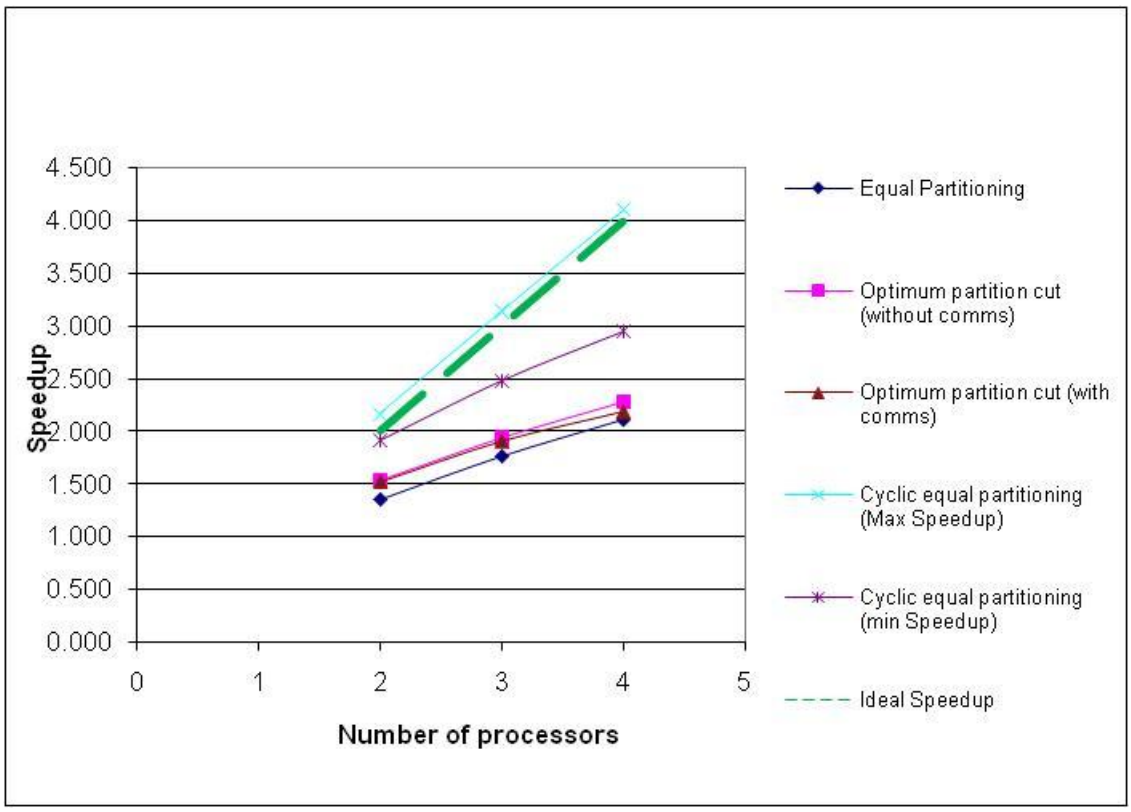



**Figure 5.20 - Comparison between the different partitioning methods used**

Using the Cyclic method has considerably improved the speedup values obtained and this is a promising approach to consider when devising the optimal partitioning method to adopt. When using the Equal Cyclic Partitioning method, better results were obtained than when the previous methods were used. The key result is that the factors that most influence the speedup values are load imbalance and processor idleness rather than communication costs.

### *5.1.4 Potential Route Map partitioning*

The Equal Cyclic Partitioning method is appropriate for cases where there is only one exit. In reality this is rarely the case; most geometries are intricate in nature and will have more than one exit.

5.1.4.1 *Potential Route Map*

In EXODUS, each geometry creates a potential route map. The Potential Route Map shows the most probable route the occupants within a region will take. When the exits have the same potential values, an illustration of how the Potential Route Map is represented is shown in Figure 5.21:

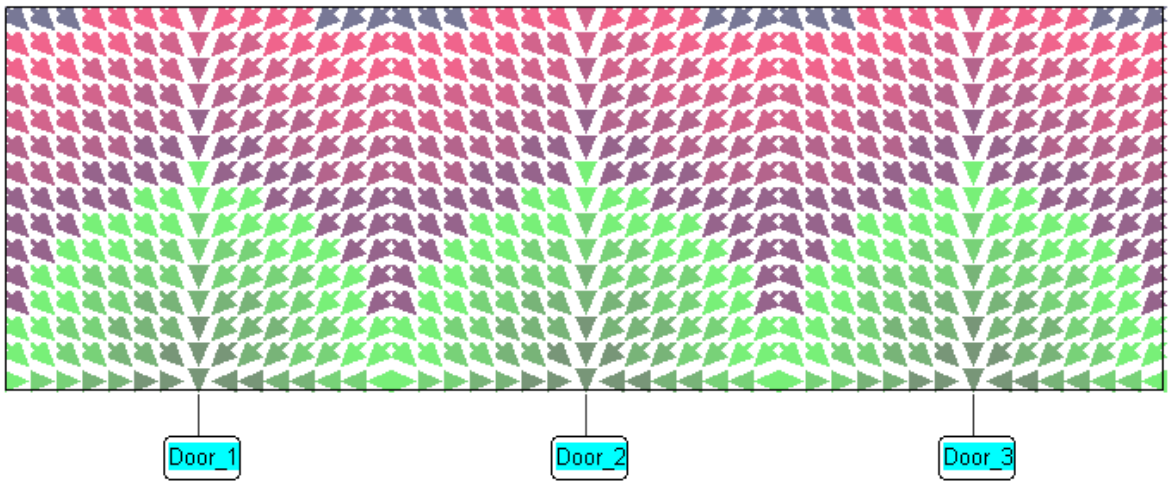



**Figure 5.21 - Potential Route Map**

Figure 5.21 represents a rectangular geometry with three exits as shown above. The arrows represent the most likely path an individual will take when in the area.

5.1.4.2 *Partitioning technique using the Potential Route Map*

The Potential Route Map automatically creates sub-regions out of the main region. The number of sub-regions created is equal to the number of available exits. The exits automatically map out their "allocated" regions, whereby an occupant in one of those regions will evacuate via their allocated exit. From Figure 5.21, where the arrows diverge, a domain can be defined and in total there are three such domains. If a partitioning boundary can be set along those diverging arrows, a natural partitioning occurs and the resulting sub-domains can be allocated to different processors. Figure 5.22 illustrates the above idea.

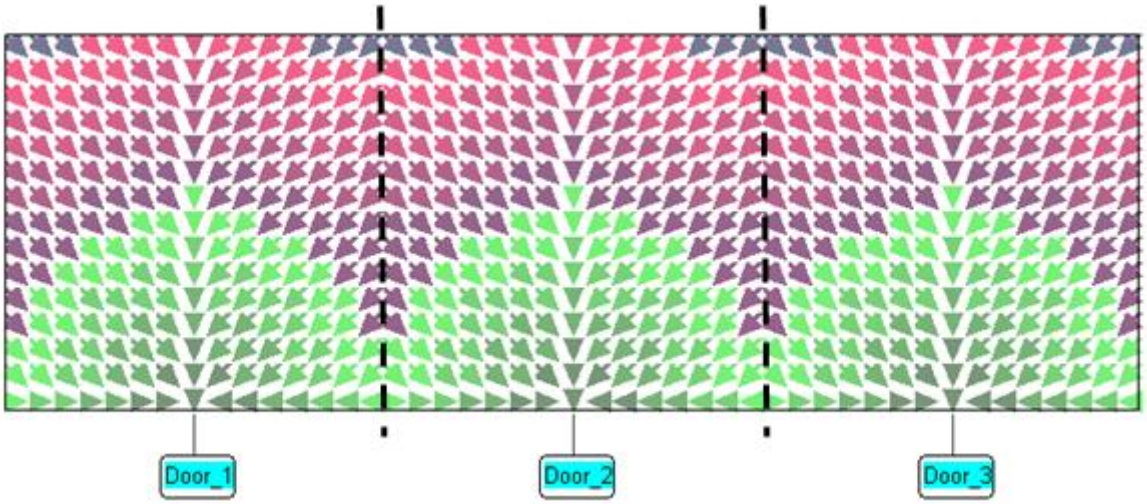



**Figure 5.22 - Example of Potential Route Map partitioning**

On both sides of these natural boundaries, it is expected that the people will diverge and hence will typically not be crossing them. Creating a partition along those boundaries seems to be the best way to minimise the communication costs since most occupants will not be crossing those boundaries.

5.1.4.3 *Results from testing the Potential Route Map partitioning method*

5.1.4.3.1 Test 1 – Potential Route Map partitioning method: Real life geometry – Trafalgar Square

As mentioned in 5.1.3, most real life geometries are irregular in design; e.g the Trafalgar Square geometry. Applying the Potential Route Map partitioning to that region demonstrates the following partitioned Trafalgar Square (when 2 and 7 processors were used) in Figure 5.23:

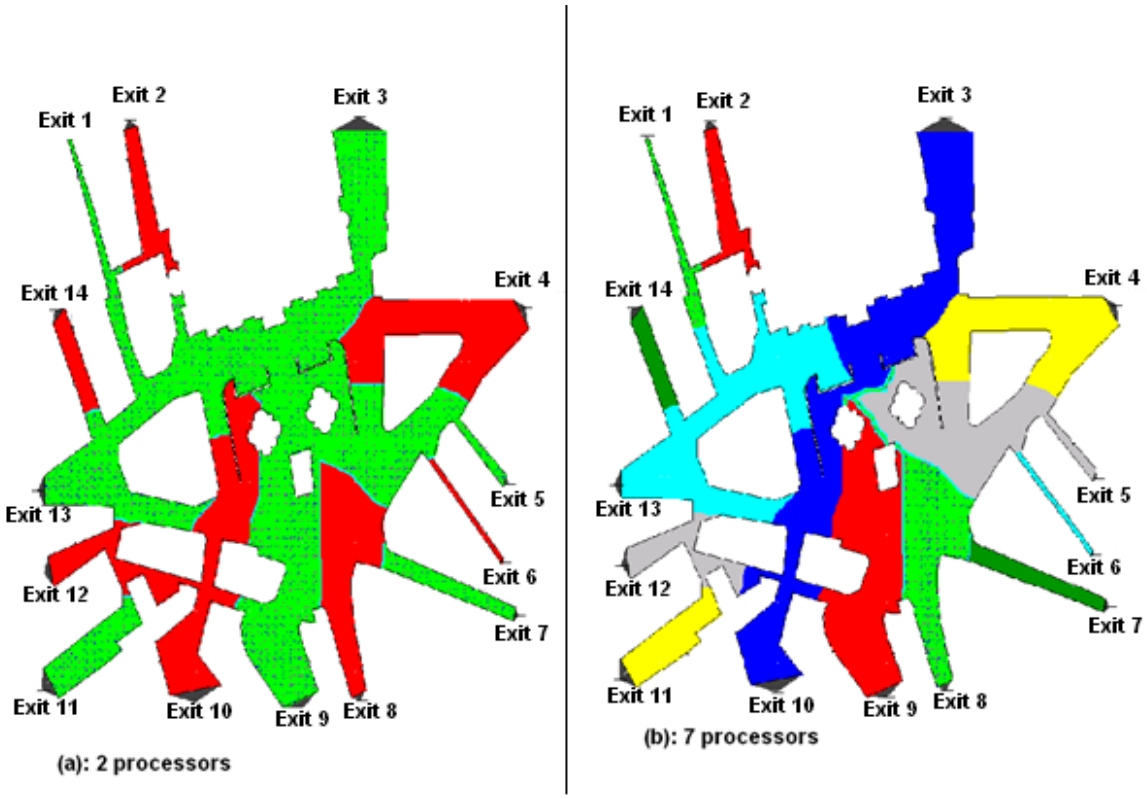



**Figure 5.23 - Trafalgar Square partitioned using the Potential Route Map partitioning method**

This geometry has 14 exit points and thus 14 sub-regions were created by the Potential Route Map partitioning method. Each sub-region (represented by a different colour) is associated to an exit point and an occupant in a sub-region will most likely evacuate via its associated exit point.

The disadvantage of this method is that the sub-regions are very different in sizes which could make the system load imbalanced and some processors may become idle whilst other processors are still working.

'Test Case 5 - Replica of the Trafalgar square in London populated by 60,000 occupants with 14 exits in total' was used to test the Potential Route Map partitioning. The predicted simulation time required to evacuate all 60,000 occupants from that geometry is 610s. Two to seven processors were used to run the case in parallel. The allocation of the sub-regions was achieved by assigning each exit point, and hence it's associated sub-region, to a different processor. When the last processor has been assigned, the next exit point was linked to the first processor again and the cycle restarted again.

For example, in the case of having two processors working on the domain (as shown in Figure 5.23 (a)), exit 1 was assigned to Processor 1, exit 2 to Processor 2 and then exit 3 back to Processor 1 and so on. The speedup and efficiency values obtained are shown in Table 5.11:

**Table 5.11 - Performance ratios over actual simulation times when the Potential Route Map partitioning was applied to the Trafalgar Square**

| Number of processors | Speedup | Efficiency | Parallel SimTime[1] Performance Ratio |
|---|---|---|---|
| **2** | 1.61 | 80.55% | 2.27 |
| **3** | 2.59 | 86.44% | 3.61 |
| **4** | 2.25 | 56.16% | 3.29 |
| **5** | 2.45 | 49.03% | 3.40 |
| **6** | 3.18 | 52.96% | 4.65 |
| **7** | 3.94 | 56.27% | 5.57 |

The speedup values in Table 5.11 do not indicate a definite improvement from the previous method used. The values obtained for up to three processors are encouraging as they are marginally less than the ideal speedup achievable. However, as the number of processors was increased, the speedup values do not increase accordingly, hence not suggesting a good scope for scalability.

From Table 5.11, the maximum performance ratio with respect to the simulation time is 5.57, not sufficiently fast to provide advice to incident commanders for live evacuations but can be an option to consider to investigate evacuation scenarios for planning and possibly interactive training purposes.

[1] **SimTime:** Simulation time (See Section 2.3.4)

The efficiency values are represented in Figure 5.24.

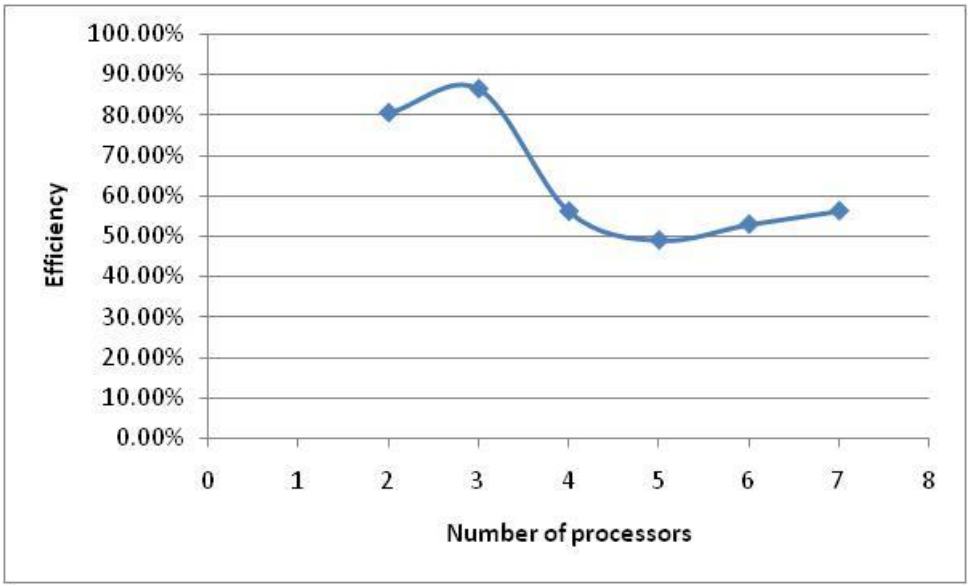



**Figure 5.24 - Efficiency graph when the Potential Route Map was applied to the Trafalgar Square**

The trend in this graph shows a decrease in efficiency as the number of processors increases. This implies that as the problem is being subdivided into more parts with more processors allocated to each part, the efficiency of the parallel system diminishes. Possible reasons for this are the increased communication and synchronisation costs or some processors being idle as their domains are people free. According to this result, adding more processing power may not necessarily produce improvements.

Figure 5.25 shows a comparison of the speedup values obtained to those from using the Equal Partitioning method.

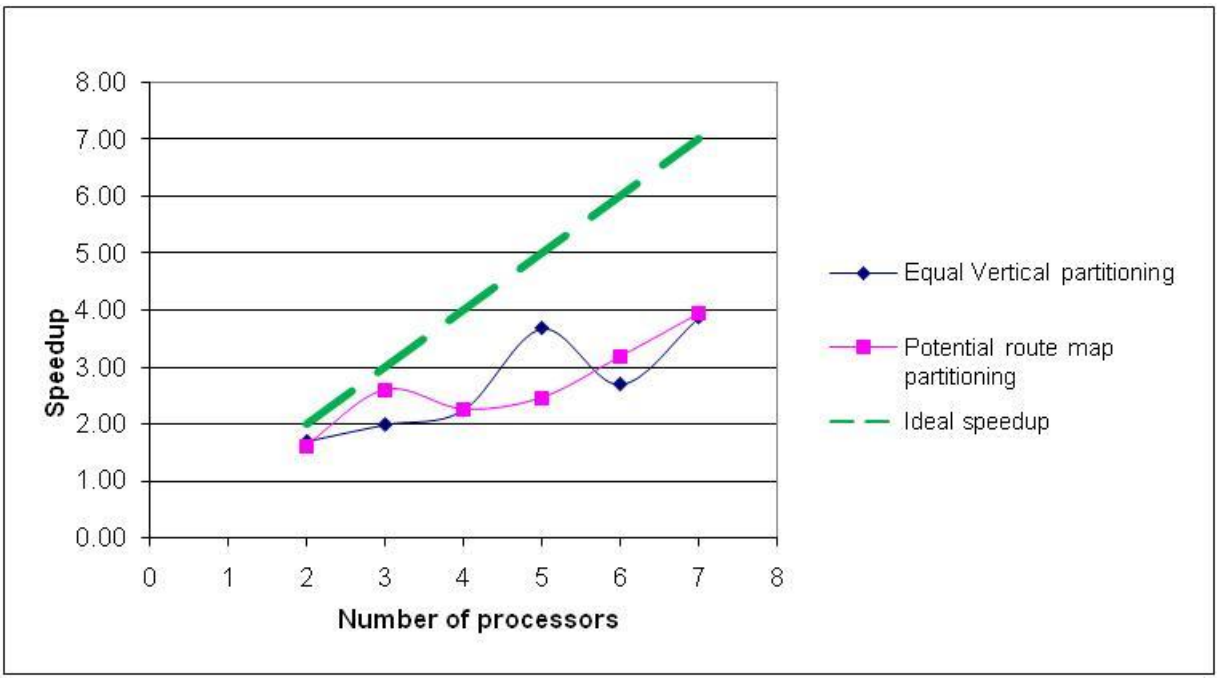



**Figure 5.25 - Comparison of speedup values obtained from using the different partitioning methods on the Trafalgar Square**

Surprisingly, the speedup values from using the Potential Route Map partitioning did not produce a marked improvement from using the Equal Partitioning method. However, the speedup values obtained shows there is some scope for scalability, i.e. as the number of processors used increases, the speedup values increase accordingly. Even with this minor increase, it is not enough to confirm that this method is useful to adopt. A method that produces speedup values close enough to the "Ideal Speedup" line would be the better method to implement.

#### 5.1.4.4 *Problems associated with the Potential Route Map partitioning method*

Though the Potential Route Map partitioning method minimises the communication costs, it does not guarantee a good load balance. As can be seen in Figure 5.23, the parts created are of different sizes which could imply a possible load imbalance. As shown in 5.1.3.3, load balancing is a more influential factor than communication costs; the Cyclic Equal Partitioning Method favours the load balance over communication costs and produced the best results achievable by producing super-linear speedup values.

Another crucial problem associated with this method is when the number of exits is less than the number of processors used. The Potential Route Map method can only create parts equal to the available exits and this is a problem when there are more processors than exits. This would suggest that the extra processor will never be used and hence the full potential of the parallel system will not be utilised. An example of this is 'Test Case 7 - Open rectangular geometry measuring 100m x 1,000m populated by 100,000 people', using the Potential Route Map partitioning on that domain will only create one sub-domain and hence no scope from using the parallel system at all. As seen in 5.1.3.3, this test case is capable of producing very encouraging results if the right partitioning method was applied to it.

### *5.1.5 Cyclic Potential Route Map partitioning*

From the previous partitioning methods attempted, some valuable findings were discovered. From the Equal Cyclic Partitioning method, it was seen that breaking a domain into smaller sub-domains tended to improve the load imbalance. The Potential Route Map partitioning automatically creates sub-domains that minimise the communication costs. Each technique had its own advantages as well as disadvantages, and merging the advantageous factors will definitely be an improvement from the previous methods attempted.

A hybrid method, merging the cyclic properties from the Equal Cyclic Partitioning method with the Potential Route Map partitioning technique, was devised. This new technique is called the Cyclic Potential Route Map (CPRM) partitioning method and can potentially solve the problems created from the Potential Route Map partitioning method. Similar to how Quinn et al [89] and the Transportation Research analysis computing centre [82] had to develop ways to minimise communication costs whilst preserving load balance, a possible solution can be obtained by the CPRM. This hybrid method should now create a load balanced problem as well as solving the problem when the number of exits is less than the number of available processors.

In theory, each exit will create its original sub-domain (where all the inhabitants of that sub-domain should be attracted to it) and then further sub-divides that sub-domain into equal parts (called sub-parts).

'Test Case 7 - Open rectangular geometry measuring 100m x 1,000m populated by 100,000 people' could not be partitioned from using the Potential Route Map partitioning method. But now using this improved technique, the resulting partitioned domain is the same as that the one obtained from using the Equal Cyclic Partitioning method which produced the best results obtained for that case.

#### 5.1.5.1 *Results from testing the Cyclic Potential Route Map partitioning method*

##### 5.1.5.1.1 Test 1 – Cyclic Potential Route Map partitioning method

Applying the Cyclic Potential Route Map partitioning to 'Test Case 5 - Replica of the Trafalgar square in London populated by 60,000 occupants with 14 exits in total', a similar test criterion was attempted. This method, as explained in Section 5.1.5, sub-partitions the partitions created by the PRM method into smaller sub-parts to promote load balance. For 60,000 occupants to physically evacuate this geometry requires a simulation time of 610s. Up to seven processors were used but different number of cyclic sub-parts at each exit. Figure 5.26 shows the geometry being partitioned by the PRM method.

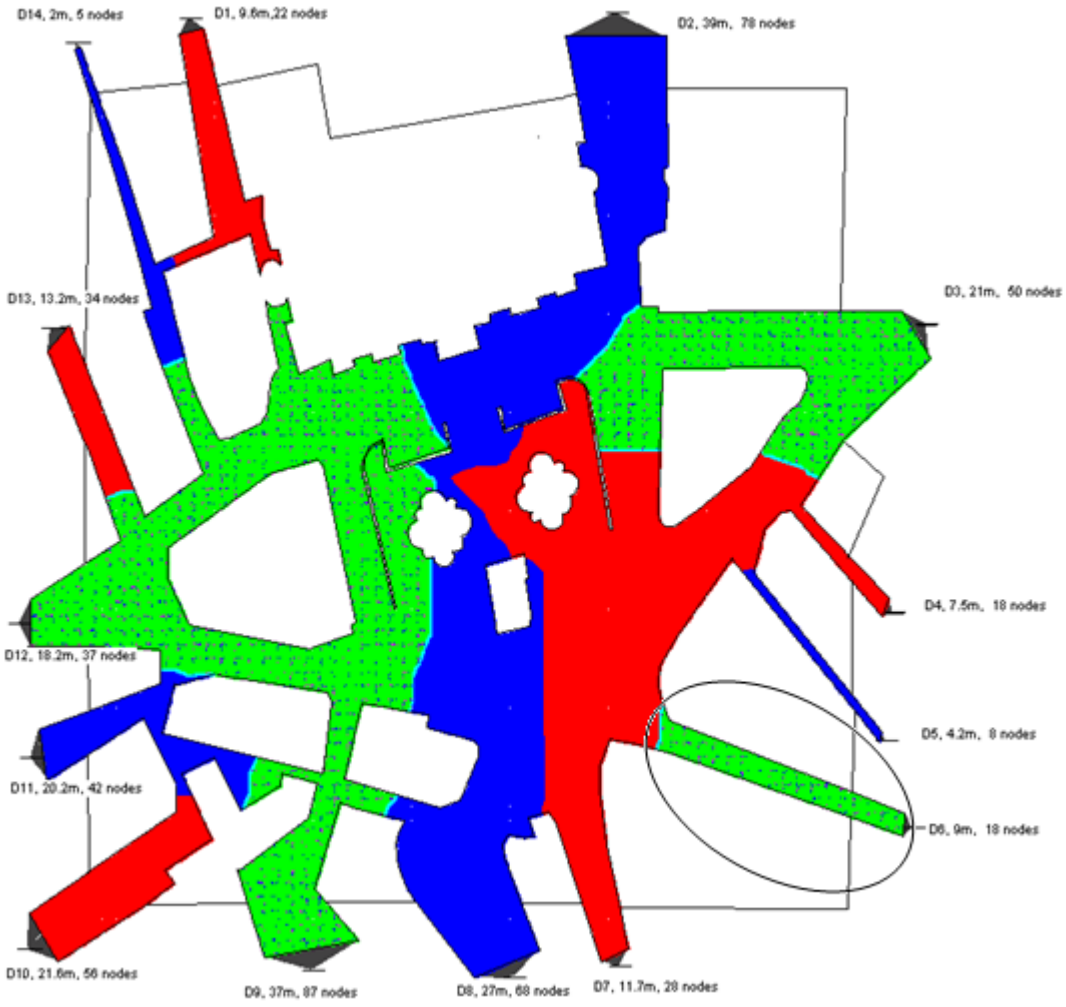



**Figure 5.26 - Trafalgar Square partitioned using the Potential Route Map partitioning showing the sub-domain created by D6**

Figure 5.26 illustrates the CPRM method being applied to the Trafalgar Square replica geometry. The highlighted section represents the partition created by the PRM method at Exit 6. This section is further sub-partitioned into 3 sub-parts as illustrated in Figure 5.27. With the CPRM method, all the original partitions created by the PRM method are being further sub-partitioned into smaller sub-parts.

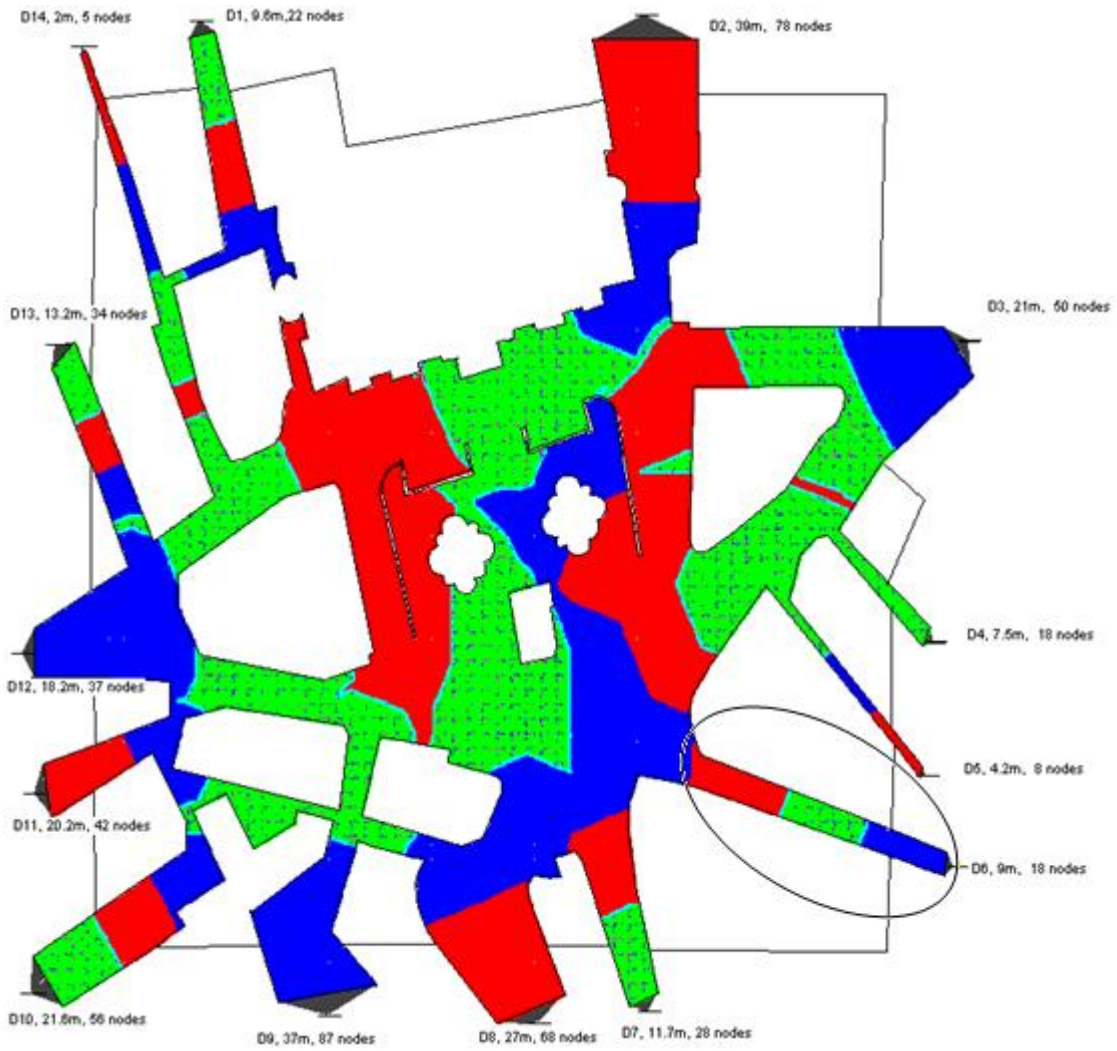



**Figure 5.27 - Trafalgar Square partitioned using the CPRM partitioning showing the cyclic sub-parts created by D6. Here three sub-parts were used to partition each sub-domain.**

The optimal number of sub-parts to create at each exit is still an unknown, so a range of number of parts were devised and tested. The same Test Case 5 was used with up to seven processors and also with dividing each exit's sub-domain into two to seven sub-parts. The results obtained are shown in Table 5.12:

**Table 5.12 - Speedup values from using the CPRM technique**

| Number of processors ↓ | Speedup values from using the CPRM partitioning | | | | | |
|---|---|---|---|---|---|---|
| **Number of sub-parts →** | **2 sub-parts** | **3 sub-parts** | **4 sub-parts** | **5 sub-parts** | **6 sub-parts** | **7 sub-parts** |
| **2** | 1.77 | 1.85 | 1.70 | 1.72 | 1.73 | 1.67 |
| **3** | 2.29 | 2.30 | 2.13 | 2.31 | 2.32 | 2.38 |
| **4** | 2.42 | 2.98 | 3.03 | 2.84 | 3.16 | 3.12 |
| **5** | 2.38 | 2.60 | 2.26 | 2.72 | 2.81 | 3.00 |
| **6** | 3.04 | 2.89 | 2.50 | 2.83 | 2.93 | 3.28 |
| **7** | 3.91 | 3.63 | 3.38 | 3.43 | 3.25 | 3.49 |

The results obtained do not give an indication of any improvement from using this method. As can be seen from Table 5.12, only a maximum speedup of 3.91 can be obtained from using seven processors. Using seven processors, a reasonable speedup value close to six is desired. Comparing the maximum speedup value (3.16) obtained from using four processors; an additional three processors could only improve the speedup marginally to 3.91, which indicates a diminishing returns as well as a waste of processor resources. Another problem might be that the regions simulated by some processors might become empty much earlier than the other regions, hence making the processors responsible for them to become idle.

Table 5.13 gives an indication on how efficiently the processors were utilised:

**Table 5.13 - Efficiency values obtained when the CPRM was used to partition Test Case 5**

| **Number of processors ↓** | **Efficiency values from using the CPRM partitioning** | | | | | |
|---|---|---|---|---|---|---|
| **Number of sub-parts →** | **2 sub-parts** | **3 sub-parts** | **4 sub-parts** | **5 sub-parts** | **6 sub-parts** | **7 sub-parts** |
| **2** | 88.50% | 92.50% | 85.00% | 86.00% | 86.50% | 83.50% |
| **3** | 76.33% | 76.67% | 71.00% | 77.00% | 77.33% | 79.33% |
| **4** | 60.50% | 74.50% | 75.75% | 71.00% | 79.00% | 78.00% |
| **5** | 47.60% | 52.00% | 45.20% | 54.40% | 56.20% | 60.00% |
| **6** | 50.67% | 48.17% | 41.67% | 47.17% | 48.83% | 54.67% |
| **7** | 55.86% | 51.86% | 48.29% | 49.00% | 46.43% | 49.86% |

When two processors were used, a maximum efficiency value of 92.5% was obtained, implying a near optimal usage of both processors. However, as the number of processors was increased, the efficiency of the parallel system became consistently lower, with three processors achieving a maximum efficiency factor of 79.33% whilst the remaining 20.67% being lost by various costs and latency. This decline got worse as extra processors were added with seven processors only achieving a maximum efficiency value of 55.86%. This implies that the seven processors are only achieving just above half of their capacity and the rest being wasted in costs and latency. Figure 5.28 is showing the maximum efficiency achieved when the Cyclic Potential Route Map technique was applied to Test Case 5.

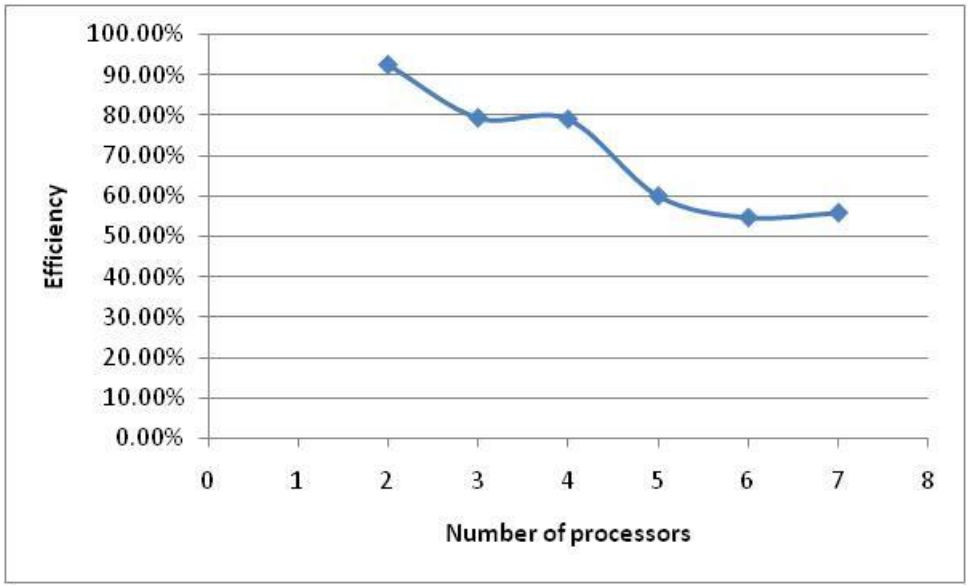



**Figure 5.28 - Maximum efficiency when the CPRM partitioning technique was used on Test Case 5**

Here again, the trend that the Efficiency graph takes suggests that adding more processors is resulting in an increasing loss of processor power. As can be seen from Figure 5.28, as the number of processors is increasing, the efficiency is declining, implying the extra processing power is not being optimally utilised.

A comparison of the Cyclic Potential Route Map partitioning method (CPRM) to that on the Potential Route Map (PRM) partitioning method is illustrated in Figure 5.29:

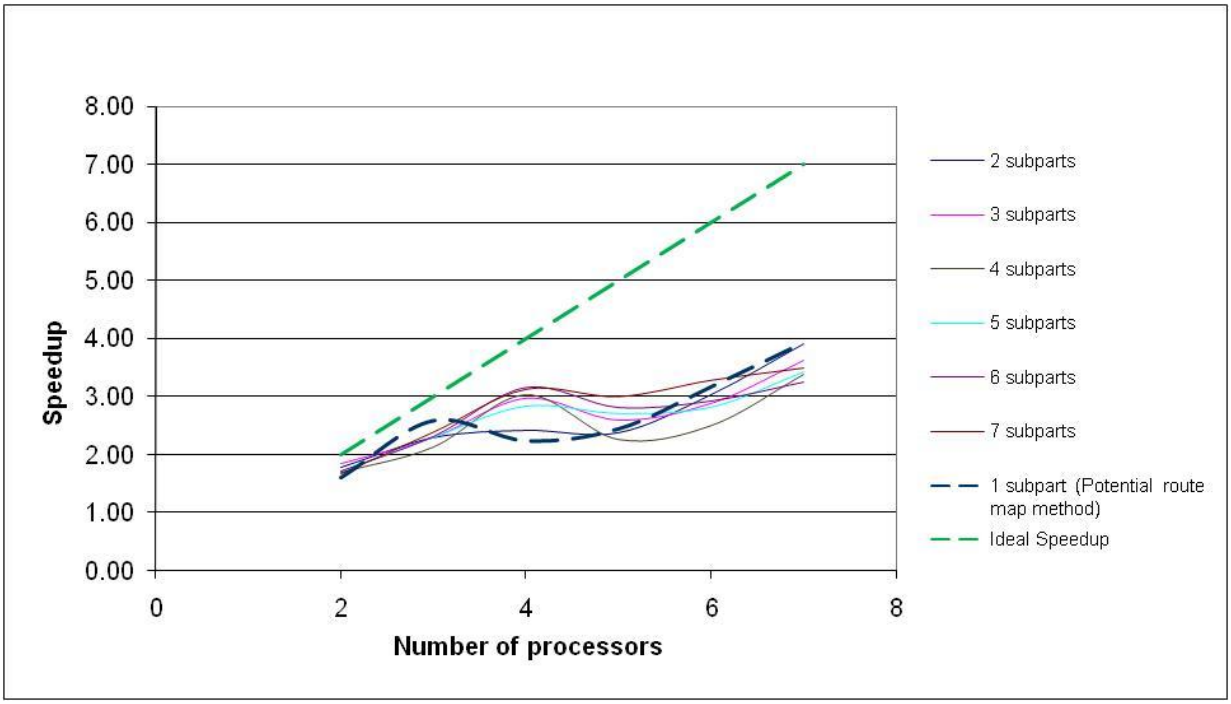



**Figure 5.29 - Comparison from using the CPRM partitioning method to that of the PRM method.**

Comparing the CPRM partitioning method to that of the PRM partitioning, whereby only one sub-part is created at each exit, no marked improvements was noticed. All the speedup values obtained do not seem to scale well as the number of processors are increased. The values should be close to the Ideal speedup line for it to show any scalability.

From the comparisons done, the optimal number of sub-parts to create at each exit is still not found. The choice of the number of sub-parts used does not have a significant impact on the speedup values obtained. Hence, no indication of how many sub-parts to create at each exit to optimise the speedup could be gained from those tests.

The only advantage from using this CPRM partitioning method compared to using the previous PRM method is that it can accommodate any type of geometry, unlike the latter method. This new method, can partition any type of domain, irrespective of the number of exits present, which the Potential Route Map cannot cope with. Furthermore, since load balance is a much more influential factor than the communication costs, the CPRM method favours this formulation due to the reduced sizes of the sub-parts.

Table 5.14 illustrates the parallel performance ratio over the simulation times.

**Table 5.14 – Performance ratios over actual simulation times when the CPRM was used to partition Test Case 5**

| **Number of processors ↓** | **Parallel SimTime[1] Performance Ratio from using the CPRM partitioning** | | | | | |
|---|---|---|---|---|---|---|
| **Number of sub-parts →** | **2 sub-parts** | **3 sub-parts** | **4 sub-parts** | **5 sub-parts** | **6 sub-parts** | **7 sub-parts** |
| **2** | 2.45 | 2.55 | 2.34 | 2.38 | 2.38 | 2.30 |
| **3** | 3.16 | 3.17 | 2.94 | 3.18 | 3.20 | 3.29 |
| **4** | 3.33 | 4.11 | 4.18 | 3.92 | 4.36 | 4.31 |
| **5** | 3.28 | 3.59 | 3.12 | 3.75 | 3.88 | 4.14 |
| **6** | 4.19 | 3.98 | 3.45 | 3.90 | 4.04 | 4.52 |
| **7** | 5.40 | 5.01 | 4.66 | 4.73 | 4.48 | 4.81 |

From Table 5.14, the maximum performance ratio with respect to the simulation time is 5.40 and yet again is not sufficiently fast enough to be used for live evacuations. However, these results can be used to investigate evacuation scenarios for planning and training purposes.

### 5.1.5.2 *Strategies considered and rejected*

Throughout this research, several ideas and strategies were considered and then were discarded as they would not have been beneficial to the outcome of this thesis. Some of the main strategies considered are explained in the following sections.

[1] **SimTime:** Simulation time (See Section 2.3.4)

5.1.5.2.1 Exits balancing strategy

In an evacuation simulation, the geometry involved usually contains several exits which can be of different sizes. It can be assumed that at any point of the simulation, a larger exit is capable of servicing more occupants than a smaller one does and if both exits are attracting roughly the same population load, the smaller one will take longer to empty.

Using the Cyclic Potential Route Map (CPRM) method to partition a geometry, the algorithm loops round the list of the exits, then loops round the list of processors and allocates each one to a processor without taking into consideration their sizes. This method does not guarantee a fair distribution of the total exits sizes.

Consider the following example in Figure 5.30 to illustrate this problem: four processors are used to simulate the evacuation of the following geometry containing eight exits, four large (D1, D2, D5 and D6) and four smaller ones (D3, D4, D7 and D8).

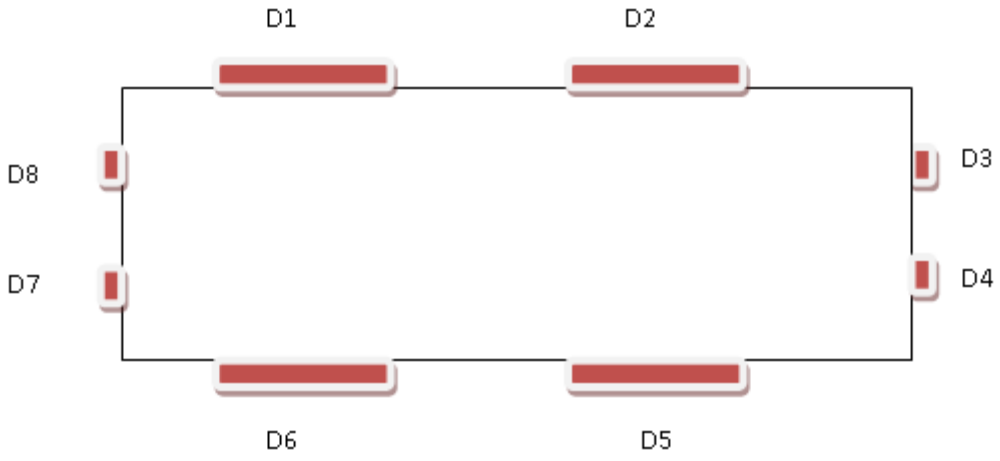



**Figure 5.30 - Geometry containing unequal sized exits**

Using the CPRM partitioning technique, the algorithm would allocate the exits in the following manner as shown in Table 5.15:

**Table 5.15 - Exits' allocation to processors by the CPRM algorithm**

| **Exit** | D1 | D2 | D3 | D4 | D5 | D6 | D7 | D8 |
|---|---|---|---|---|---|---|---|---|
| **Size** | Large | Large | Small | Small | Large | Large | Small | Small |
| **Processor** | P1 | P2 | P3 | P4 | P1 | P2 | P3 | P4 |

Table 5.15 demonstrates how the CPRM partitioning algorithm does not guarantee a fair distribution of the exits according to their sizes. Ideally each processor should get two exits, a large and a small one.

The CPRM algorithm was modified to include an exit width balancing criteria. The exits are sorted according to their widths and the algorithm then allocates them in such a way that each processor gets roughly an equal portion of the total size of all the exits. This algorithm was tested on Test Case 5 - Replica of the Trafalgar square in London populated by 60,000 occupants with 14 exits in total' and demonstrates the changes made.

The processors are colour coded to ease the representation of the exits on the different processors. An illustration is shown below:

Legends:

Colour representation of processors


Table 5.16 illustrates the distribution of the exits before the exit width balancing algorithm was applied.

**Table 5.17 - Distribution of the exits before the exit width balancing algorithm was applied**

| | 2 processors | 3 processors | 4 processors | 5 processors | 6 processors | 7 processors |
|---|---|---|---|---|---|---|
| **Exits** | **Size(nodes)** | **Size(nodes)** | **Size(nodes)** | **Size(nodes)** | **Size(nodes)** | **Size(nodes)** |
| **1** | 22 | 22 | 22 | 22 | 22 | 22 |
| **2** | 78 | 78 | 78 | 78 | 78 | 78 |
| **3** | 56 | 56 | 56 | 56 | 56 | 56 |
| **4** | 18 | 18 | 18 | 18 | 18 | 18 |
| **5** | 8 | 8 | 8 | 8 | 8 | 8 |
| **6** | 18 | 18 | 18 | 18 | 18 | 18 |
| **7** | 29 | 29 | 29 | 29 | 29 | 29 |
| **8** | 70 | 70 | 70 | 70 | 70 | 70 |
| **9** | 85 | 85 | 85 | 85 | 85 | 85 |
| **10** | 56 | 56 | 56 | 56 | 56 | 56 |
| **11** | 43 | 43 | 43 | 43 | 43 | 43 |
| **12** | 37 | 37 | 37 | 37 | 37 | 37 |
| **13** | 36 | 36 | 36 | 36 | 36 | 36 |
| **14** | 5 | 5 | 5 | 5 | 5 | 5 |
| **Total exit width** | **561** | **561** | **561** | **561** | **561** | **561** |

As can be seen from Table 5.17, the old algorithm balanced the number of exits rather than the total exits widths.

Table 5.18 shows the percentage of the total exits widths allocated to each processor:

**Table 5.18 - Percentage of the total exits widths allocated to each processor**

| | 2 processors | 3 processors | 4 processors | 5 processors | 6 processors | 7 processors |
|---|---|---|---|---|---|---|
| P0 | 49.73% | 34.94% | 22.28% | 11.41% | 9.80% | 6.06% |
| P1 | 50.27% | 28.70% | 26.92% | 14.80% | 15.51% | 16.40% |
| P2 | | 36.36% | 27.99% | 25.67% | 27.27% | 29.06% |
| P3 | | | 22.82% | 28.88% | 25.13% | 19.96% |
| P4 | | | | 19.25% | 13.19% | 10.87% |
| P5 | | | | | 9.09% | 8.02% |
| P6 | | | | | | 9.63% |

Table 5.18 again demonstrates the unequal distribution in exit widths when the old algorithm was applied. The modified exit width balancing algorithm was then applied to the same test case and the following exits distribution (shown in Table 5.19) is produced:

**Table 5.19 - Distribution of the exits after the exit width balancing algorithm was applied**

| | 2 processors | 3 processors | 4 processors | 5 processors | 6 processors | 7 processors |
|---|---|---|---|---|---|---|
| **Exits** | **Size(nodes)** | **Size(nodes)** | **Size(nodes)** | **Size(nodes)** | **Size(nodes)** | **Size(nodes)** |
| **1** | 22 | 22 | 22 | 22 | 22 | 22 |
| **2** | 78 | 78 | 78 | 78 | 78 | 78 |
| **3** | 56 | 56 | 56 | 56 | 56 | 56 |
| **4** | 18 | 18 | 18 | 18 | 18 | 18 |
| **5** | 8 | 8 | 8 | 8 | 8 | 8 |
| **6** | 18 | 18 | 18 | 18 | 18 | 18 |
| **7** | 29 | 29 | 29 | 29 | 29 | 29 |
| **8** | 70 | 70 | 70 | 70 | 70 | 70 |
| **9** | 85 | 85 | 85 | 85 | 85 | 85 |
| **10** | 56 | 56 | 56 | 56 | 56 | 56 |
| **11** | 43 | 43 | 43 | 43 | 43 | 43 |
| **12** | 37 | 37 | 37 | 37 | 37 | 37 |
| **13** | 36 | 36 | 36 | 36 | 36 | 36 |
| **14** | 5 | 5 | 5 | 5 | 5 | 5 |
| **Total exit width** | **561** | **561** | **561** | **561** | **561** | **561** |

From Table 5.19, it can now be seen that the algorithm is no longer allocating an equal number of exits to each processor but is now allocating the exits based on their widths.

Table 5.20 demonstrates the percentage of the exits widths allocated to each processor.

**Table 5.20 - Percentage of the total exits widths allocated to each processor**

| | 2 processors | 3 processors | 4 processors | 5 processors | 6 processors | 7 processors |
|---|---|---|---|---|---|---|
| P0 | 50.09% | 33.33% | 24.78% | 19.07% | 16.04% | 15.15% |
| P1 | 49.91% | 33.69% | 25.13% | 19.96% | 17.11% | 13.90% |
| P2 | | 32.98% | 24.96% | 20.32% | 16.40% | 15.69% |
| P3 | | | 25.13% | 19.79% | 16.58% | 13.19% |
| P4 | | | | 20.86% | 16.40% | 13.90% |
| P5 | | | | | 17.47% | 14.26% |
| P6 | | | | | | 13.90% |

This new allocation of the total exits' width is much fairer than the old algorithm was formulating. Each processor now has roughly the same total exit widths instead of the same number of exits which does not guarantee a balance in the total exit widths.

Before adopting this new algorithm, it was tested by running the simulations to see if there were any significant improvements. The CPRM method sub-partitioned each sub-domain at each exit into a number of sub-parts (1-6). The predicted simulation time to evacuate a population of 60,000 for this geometry is 610s. The results are illustrated in Table 5.21:

**Table 5.21 - Speedup values obtained when both the old and new exit widths balancing algorithm was applied to Test Case 5**

| | 2 processors | | 3 processors | | 4 processors | |
|---|---|---|---|---|---|---|
| | Speedup | | Speedup | | Speedup | |
| **No. of sub-parts** | **Before** | **After** | **Before** | **After** | **Before** | **After** |
| **1** | 1.61 | 1.62 | 2.59 | 2.26 | 2.25 | 2.37 |
| **2** | 1.77 | 1.60 | 2.29 | 2.24 | 2.42 | 2.80 |
| **3** | 1.85 | 1.64 | 2.30 | 2.18 | 2.98 | 2.57 |
| **4** | 1.70 | 1.70 | 2.13 | 2.35 | 3.03 | 2.68 |
| **5** | 1.72 | 1.75 | 2.31 | 2.33 | 2.84 | 3.03 |
| **6** | 1.73 | 1.70 | 2.32 | 2.33 | 3.16 | 2.90 |

The tests run were abandoned in the early stages of testing as no definite improvements were gained from the exit widths balancing algorithm. The speedup values obtained were similar when both algorithms were used, indicating that the new algorithm does not produce an obvious advantage. This new algorithm contains a sorting algorithm which is an extra cost to incur, and since it was not a deterministic success, the idea of balancing exit widths amongst the processors was rejected.

#### 5.1.5.2.2 Partitioning software refining original partitions

The partitions created by the Cyclic Potential Route Map (CPRM) method can be of unbalanced shapes and sizes. These partitions sometimes are too thin in sizes or too small compared to the other partitions, which reduces the efficiency of a parallel performance due to added communications and load imbalance. An example of such a problem is highlighted by the oval outlines around the thin and small partitions as shown in Figure 5.31.

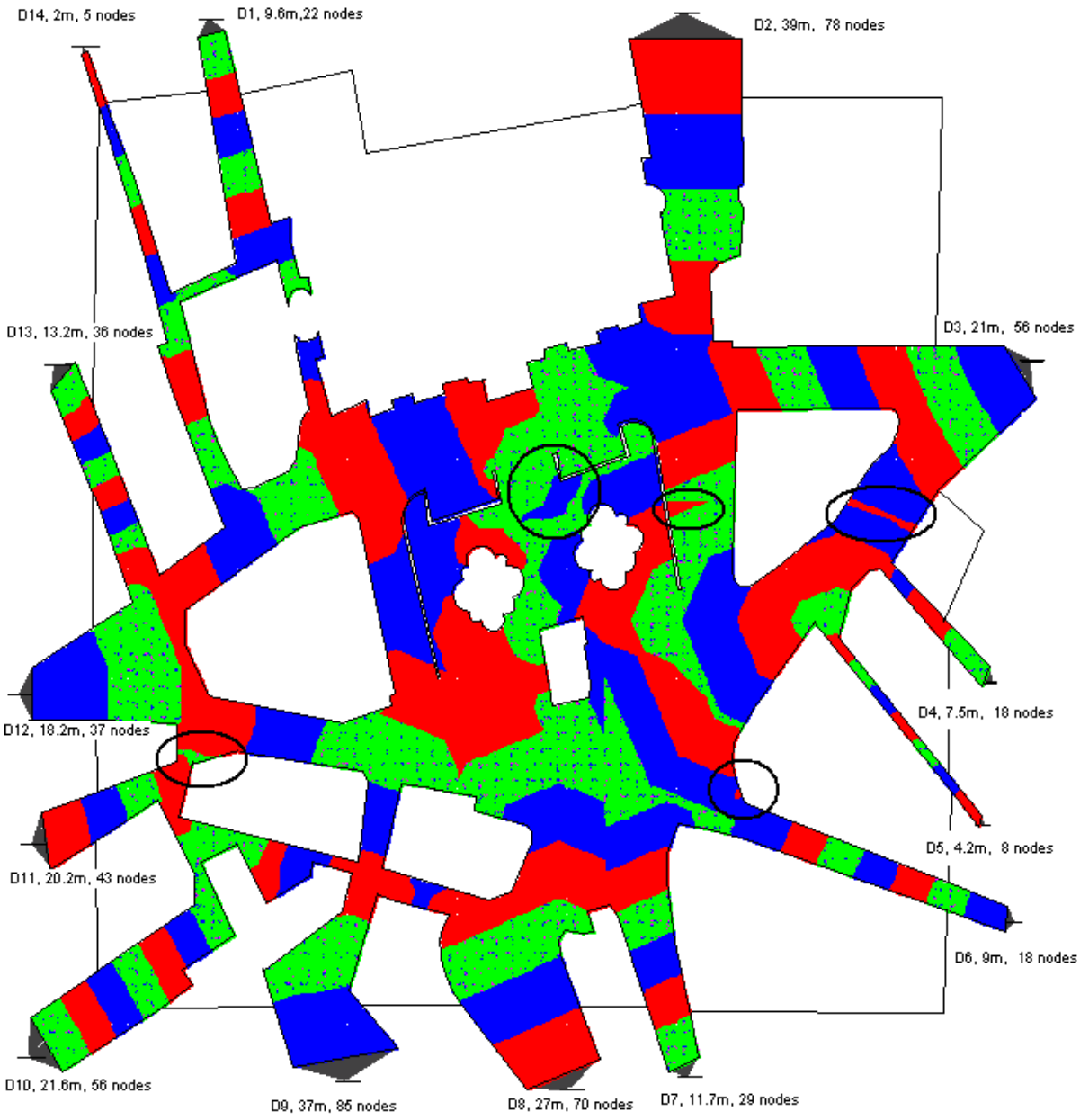



**Figure 5.31- Thin and small partitions created by the CPRM method on Test Case 5**

The partitioning software JOSTLE [124] has the capability to refine the partitions created by a partitioning method. Hence, JOSTLE was used on Test Case 5 to refine the original partitions created by the CPRM method and the refined partitions are shown in Figure 5.32.

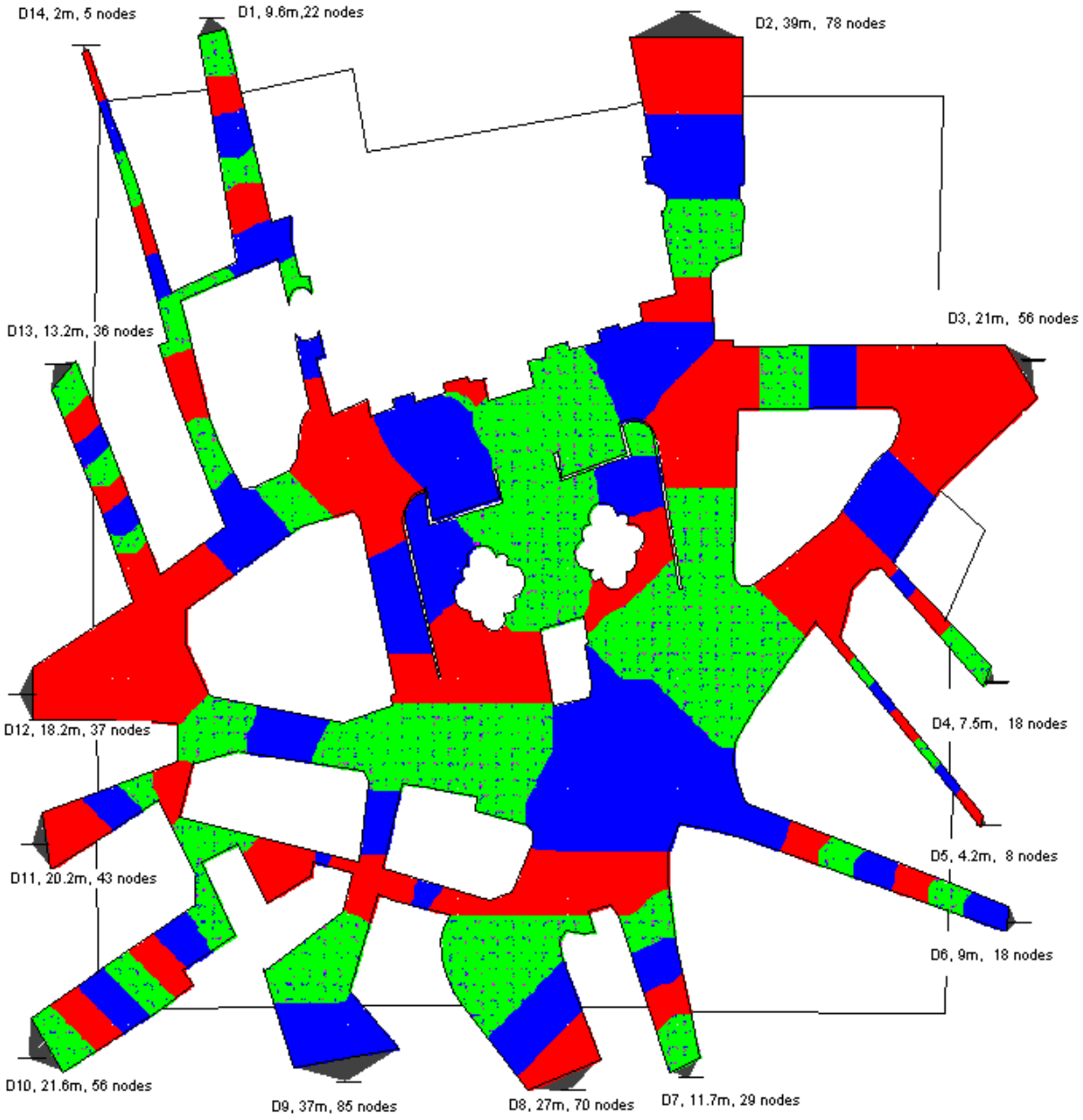



**Figure 5.32 - JOSTLE refining the original partitions of Test Case 5 set by the CPRM method**

Figure 5.32 shows a significant reduction in the number of thin and small partitions that was created by the CPRM method. Hence, JOSTLE has achieved the intention to refine the original partitions. This refined problem was tested to assess whether this approach significantly improves the results obtained from using the partitions created by the CPRM method. The predicted

simulation time to evacuate a population of 60,000 from this geometry is 610s. The results obtained are displayed in Table 5.22.

**Table 5.22 - Comparisons in the speedup values obtained with and without JOSTLE's refinement**

| Number of processors | | Speedup values | | | | | |
|---|---|---|---|---|---|---|---|
| | Number of sub-parts → | 2 | 3 | 4 | 5 | 6 | 7 |
| 2 processors | CPRM (with JOSTLE refinement) | 1.65 | 1.73 | 1.86 | 1.84 | 1.61 | 1.74 |
| | CPRM (without JOSTLE refinement) | 1.77 | 1.85 | 1.70 | 1.72 | 1.73 | 1.67 |
| 3 processors | CPRM (with JOSTLE refinement) | 2.65 | 2.26 | 2.07 | 2.02 | 2.04 | 2.12 |
| | CPRM (without JOSTLE refinement) | 2.29 | 2.30 | 2.13 | 2.31 | 2.32 | 2.39 |
| 5 processors | CPRM (with JOSTLE refinement) | 4.02 | 3.54 | 2.16 | 2.47 | 2.59 | 2.44 |
| | CPRM (without JOSTLE refinement) | 2.38 | 2.60 | 2.26 | 2.72 | 2.81 | 3.00 |

From the results in Table 5.22, there was not a significant difference in the speedup values obtained whenever JOSTLE was used to refine the partitions. In the majority of cases, better speedup values were obtained whenever the CPRM method was used without any refinement from JOSTLE. Hence, an alternative scheme is needed to boost up the speedup values as this method did not improve the results considerably.

Furthermore, JOSTLE [146] is freely available for academic and research purposes and a licence agreement is needed for commercial software. buildingEXODUS, being a commercial software, needs to obtain a licence for JOSTLE before employing it. If there was a significant

improvement on the results obtained, then JOSTLE would have been a strong option to consider. On the other hand, METIS [122, 147, 148] is freely available to use by any users and consequently provides an easier access to it as there no requirements to obtain a licence agreement. However, METIS still needs to demonstrate that it can provide satisfactory partitions and hence provide good speedup values to qualify as a probable partitioning software to use.

#### 5.1.5.2.3 Results from using JOSTLE and METIS

Both partitioning software, JOSTLE and METIS were used on Test Case 5 as a basis for a direct comparison. The main goal of this test is to assess how METIS measures up to JOSTLE, since they both use similar partitioning techniques; see Section 2.7.1. The results obtained are shown in Table 5.23.

**Table 5.23 - Comparison of results from using JOSTLE and METIS on Test Case 5**

| | **Speedup** | | |
|---|---|---|---|
| | **2 Processors** | **3 Processors** | **5 Processors** |
| **JOSTLE** | 1.71 | 2.50 | 3.60 |
| **METIS** | 1.85 | 2.90 | 3.80 |

From the results obtained, the speedup values obtained from using METIS were consistently better than the speedups obtained from using JOSTLE. Hence, METIS is a possible option to consider as a partitioning strategy and the use of JOSTLE was abandoned.

5.1.5.2.4 Alternative Partitioning Strategies

Some speculated partitioning strategies were thought of but could not be adopted for various reasons. The two main strategies are explained in the following sections.

#### 5.1.5.2.4.1 Partitions based on population density

A straightforward partitioning strategy is to divide the domain based on its population density. Partitions will be created wherever the population density is high in a region. This strategy was not even attempted as these highly populated areas do not stay in these conditions for long. As soon as the simulation starts, the population starts dispersing and hence leaving their allocated partitions, thus rendering them with a lesser load as the simulation progresses.

#### 5.1.5.2.4.2 Initial serial run to assess movement trends to then partition accordingly

Another possible partitioning strategy is to perform a simulation in serial and then observe the paths and evacuation dynamics of the simulation. These observations can then be used to formulate a partitioning strategy based on the routes taken by the population. However, this option will not be of any benefit in an emergency scenario where time is the essence, as a serial run needs to be done in advance which deflects the whole purpose of this research.

5.1.5.2.5 Test 2 – Cyclic Potential Route Map partitioning method

Before discarding the CPRM method which seemed the most reasonable technique to adopt, another set of tests was devised and tested in the next section.

**Test Case 8 - Replica of Beijing High Street randomly populated by 50,000 occupants with nine exits in total**

Up to 10 processors were used and each exit having up to six sub-parts. There are nine exits in total. The average predicted simulation time to evacuate a population of 50,000 from this geometry is 1571.83s.

The tests run was for the Potential Route Map partitioning technique (1 sub-part at each exit) as shown in Figure 5.33(a) and by using the Cyclic Potential Route Map technique with each exit sub-partitioning the Potential Route Map partition into two to six sub-parts. Figure 5.33(b) illustrates the Cyclic Potential Route Map by using 4 sub-parts per exit.

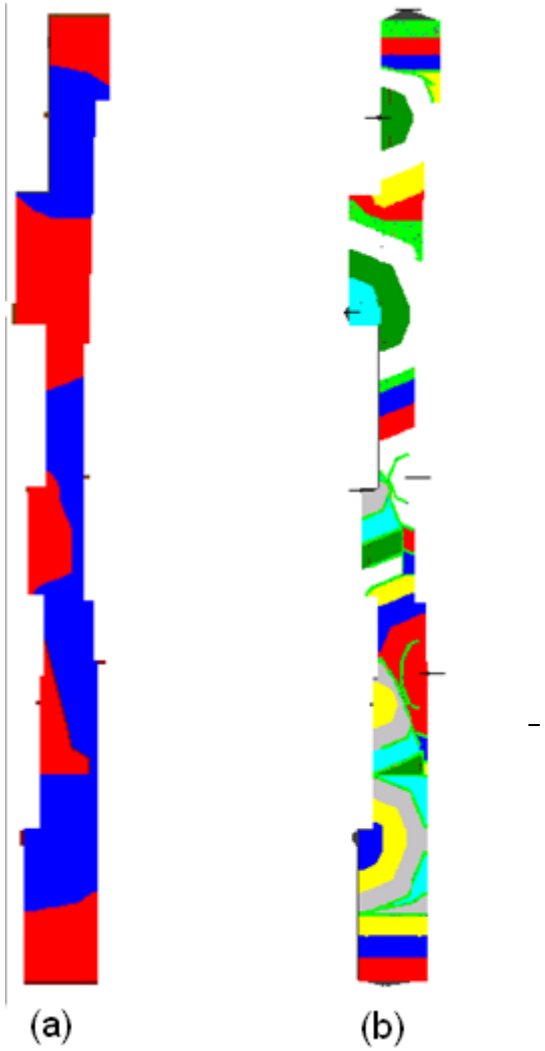



**Figure 5.33 – Replica of the Beijing High Street, partitioned using (a) the Potential Route Map method (1 sub-part at each exit) and (b) the CPRM method with 4 sub-parts at each exit**

The results obtained from running those tests are shown in Table 5.24:

**Table 5.24- Speedup values obtained from running Test Case 8**

| **Number of processors ↓** | **Speedup values from using the CPRM partitioning** | | | |
|---|---|---|---|---|
| **No. of sub-parts →** | **Potential Route Map (1 sub-part)** | **2 sub-parts** | **4 sub-parts** | **6 sub-parts** |
| **2** | 1.622 | 1.797 | 1.514 | 1.441 |
| **4** | 2.064 | 2.719 | 2.307 | 2.331 |
| **6** | 2.755 | 3.570 | 3.118 | 3.378 |
| **8** | 3.468 | 3.968 | 3.530 | 3.378 |
| **10** | 3.379 | 4.011 | 3.578 | 3.785 |

A similar conclusion can be deduced from testing the Cyclic Potential Route Map on the Beijing High street model to that from testing on the Trafalgar Square replica; using two processors produced a maximum speedup value of 1.797 which can be deemed acceptable from the ideal value of 2 by considering the costs associated from using a parallel system. However, as the number of processors was increased, the speedup values did not increase linearly enough to be close to the ideal speedup line. A maximum speedup value of 4.011 was obtained from using ten processors which is a poor parallel performance. The difference between the maximum speedup (3.570) obtained from using six processors does not differ much from the maximum value of 4.011 obtained from using ten processors. This suggests that using the capabilities of four extra processors only produced a meagre increase in the final results obtained, hence not fully utilising all these computing power that these four processors could bring. Again, the reason for this lack of improvement in the speedup values might be the fact that some processors are becoming idle if their sub-domains are emptying much earlier than the other sub-domains.

Table 5.25 shows the performance of the processors used:

**Table 5.25 - Efficiency values when the CPRM technique was applied to Test Case 8**

| **Number of processors ↓** | **Efficiency values from using the CPRM partitioning** | | | |
|---|---|---|---|---|
| **No. of sub-parts →** | **PRM (1 sub-part)** | **2 sub-parts** | **4 sub-parts** | **6 sub-parts** |
| **2** | 81.10% | 89.85% | 75.70% | 72.05% |
| **4** | 51.60% | 67.98% | 57.68% | 58.28% |
| **6** | 45.92% | 59.50% | 51.97% | 56.30% |
| **8** | 43.35% | 49.60% | 44.13% | 42.23% |
| **10** | 33.79% | 40.11% | 35.78% | 37.85% |

Using two processors achieved a maximum efficiency factor of 89.85% implying that both processors were utilised efficiently. Adding more processors unfortunately decreases the efficiency of the parallel system, with ten processors only achieving a maximum efficiency of 40.11%. Less than half the processor resources were efficiently used with the rest being either spent in communication and/or synchronisation and/or being idle.

Figure 5.34 illustrates this decline in efficiency as the number of processors is increased:

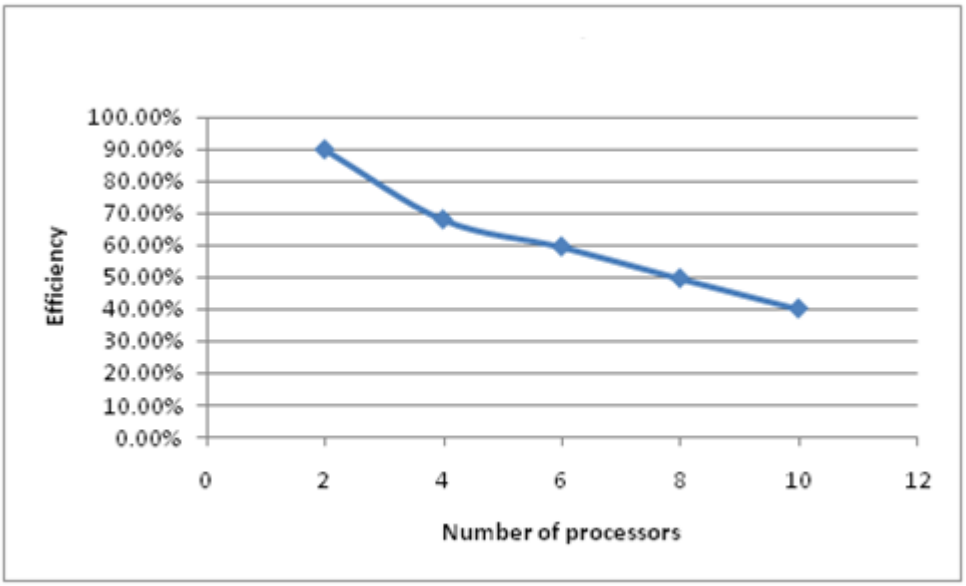



**Figure 5.34 - Maximum efficiency obtained when the CPRM technique was applied to Test Case 8**

The parallel simulation time performance ratios from using the CPRM partitioning technique are displayed in **Error! Reference source not found.**.

**Table 5.26 - Performance ratios over actual simulation times when the CPRM technique was applied to Test Case 8**

| **Number of processors ↓** | **Parallel SimTime[1] Performance ratio from using the CPRM partitioning** | | | |
|---|---|---|---|---|
| **No. of sub-parts →** | **PRM (1 sub-part)** | **2 sub-parts** | **4 sub-parts** | **6 sub-parts** |
| **2** | 3.753 | 4.158 | 3.505 | 3.334 |
| **4** | 4.776 | 6.293 | 5.339 | 5.393 |
| **6** | 6.377 | 8.262 | 7.215 | 7.784 |
| **8** | 8.025 | 9.207 | 8.170 | 7.817 |
| **10** | 7.820 | 9.281 | 8.281 | 8.759 |

[1] **SimTime:** Simulation time (See Section 2.3.4)

From Table 5.26, the maximum performance ratio with respect to the simulation time is an encouraging 9.281. Whilst an improvement over the previous values is obtained, it is still not sufficiently fast to provide advice to incident commanders for live evacuations. Nonetheless, this performance can still be usefully applied to investigate evacuation scenarios for planning purposes.

To view the differences in the values obtained from using the different number of sub-parts at each exit, the comparisons are illustrated in Figure 5.35.

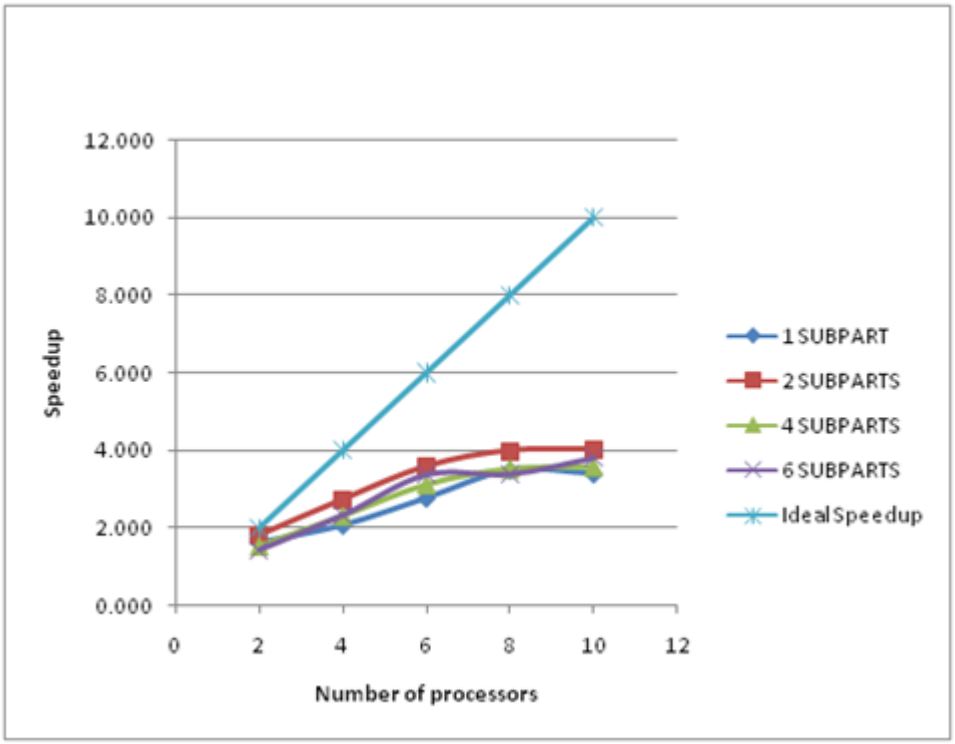



**Figure 5.35 - Speedup values obtained from using different number of sub-parts at each exit**

Test Case 8 did not produce any improvement from the results obtained from using the previous partitioning techniques. The speedup values are still too low compared to the ideal speedup achievable. The lines suggest a gradual increasing trend in speedup values until eight processors were used, then it plateaus from then on, which is a sign of poor scalability.

The only consistent trend in these results is the fact that using two sub-parts at each exit produced better results with any number of processors used. The previous tests did not give a definite number of sub-parts which consistently produced better speedup values.

### *5.1.6 Partitioning software: METIS*

Since none of the partitioning methods attempted so far did not give satisfactory parallel results, the partitioning software METIS was used to see if there are any improvements in the speedup values obtained so far. The partitioning software METIS was chosen to partition the problems.

METIS is a set of serial programs for partitioning graphs and produces partitions that are consistently better than those produced by other widely used algorithms [149]. This software tries to optimise the partitions in such a way that both load balancing and the minimisation of communication costs are preserved. As explained in Section 5.1.5.2.2, METIS is a freely available software to all users provided the papers referenced at [122, 147] are provided.

#### 5.1.6.1 *Results from testing METIS partitioning*

For a consistent comparison to the previous partitioning methods attempted so far, the same test cases were used to test the partitioning technique provided by METIS.

##### 5.1.6.1.1 Test 1 – METIS partitioning

'Test Case 5 - Replica of the Trafalgar square in London populated by 60,000 occupants with 14 exits in total' was used to test the partitioning that METIS can provide. The average predicted simulation time to evacuate a population of 60,000 from this geometry is 610s The number of sub-domains was set to vary from equalling the number of available processors used to up to 100 sub-domains. The results obtained are tabulated in Table 5.27.

**Table 5.27 - Speedup values obtained from using METIS on Test Case 5**

| Number of processors ↓ | Speedup values from using the METIS partitioning | | | | |
|---|---|---|---|---|---|
| **No. of sub-domains →** | **P[1] sub-domains** | **15 sub-domains** | **25 sub-domains** | **60 sub-domains** | **100 sub-domains** |
| **2** | 1.85 | 1.82 | 1.78 | 1.59 | 1.41 |
| **3** | 2.90 | 2.47 | 2.38 | 2.40 | 1.99 |
| **4** | 3.66 | 2.77 | 2.83 | 2.59 | 2.28 |
| **5** | 3.80 | 3.72 | 3.25 | 2.94 | 2.75 |
| **6** | 3.91 | 3.97 | 3.89 | 3.82 | 2.83 |
| **7** | 3.42 | 4.35 | 3.89 | 3.84 | 3.49 |

Using different number of sub-domains in METIS produced varying speedup values. From these values it is noticed that the choice of the number of partitions is a dependent factor to consider in obtaining the best speedup values. From Table 5.27, the trend is to divide the geometry into the number of processors available when only a small number of processors were used. When up to five processors were used, the best speedup values are from partitioning the geometry into the number of processors available. However, as the number of processors were increased (i.e 6 and 7 processors used), the best speedup values are from partitioning the geometry into 15 sub-domains. From the values obtained when 100 sub-domains were used, the impact of selecting a sub-optimal number of sub-domains can drastically deteriorate the speedup values obtainable. A poor choice of partitioning technique can lead to load imbalance; consider the speedup values obtained when seven processors were used, dividing the domain into NumProcs (Number of processors) sub-domains actually gave a worst speedup value then when the geometry was partitioned into 100 sub-domains. Again this emphasises that load imbalance and idleness are more detrimental to the results than increased communication costs. Hence, the number of sub-domains to partition the geometry needs to be carefully devised so as to prevent these

[1] p: number of processors used (e.g 2, 3, …, 7)

unfavourable occurrences. METIS produced favourable speedup values when the number of processors used are four and below; with two processors giving a speedup value of 1.85, three processors producing a 2.9 speedup and four processors giving a 3.66 speedup value. As the number of processors were increased, the resulting speedup values did not increase accordingly; with five processors producing a maximum speedup value of 3.80, six processors only producing a 3.97 speedup and seven processors only achieving 4.35.

The efficiency of the parallel system is shown in Table 5.28:

**Table 5.28 - Efficiency values from using METIS on Test Case 5**

| **Number of processors** ↓ | **Efficiency values from using the METIS partitioning** | | | | |
|---|---|---|---|---|---|
| **No. of sub-domains** → | **p[1] sub-domains** | **15 sub-domains** | **25 sub-domains** | **60 sub-domains** | **100 sub-domains** |
| **2** | 92.32% | 91.08% | 88.97% | 79.69% | 70.37% |
| **3** | 96.73% | 82.17% | 79.36% | 79.93% | 66.45% |
| **4** | 91.58% | 69.20% | 70.69% | 64.81% | 56.99% |
| **5** | 76.07% | 74.32% | 65.08% | 58.71% | 54.94% |
| **6** | 65.20% | 66.14% | 64.86% | 63.73% | 47.18% |
| **7** | 48.82% | 62.14% | 55.55% | 54.79% | 49.86% |

From the speedup values obtained, the trend suggested a good performance when up to four processors were used, with more than 90% of efficiency was achieved. This suggests that only about 10% of the processor usage was spent in parallelisation costs and the rest was usefully employed in running the simulation. However, the efficiency of the parallel system deteriorated after more processors were added, which implies that communication, synchronisation costs and idleness became a more important factor affecting the system whereby the processors spent more

[1] p: number of processors used (e.g 2, 3, …, 7)

time in dealing with those costs than when less than four processors were used. Figure 5.36 displays the maximum efficiency obtained from Table 5.28.

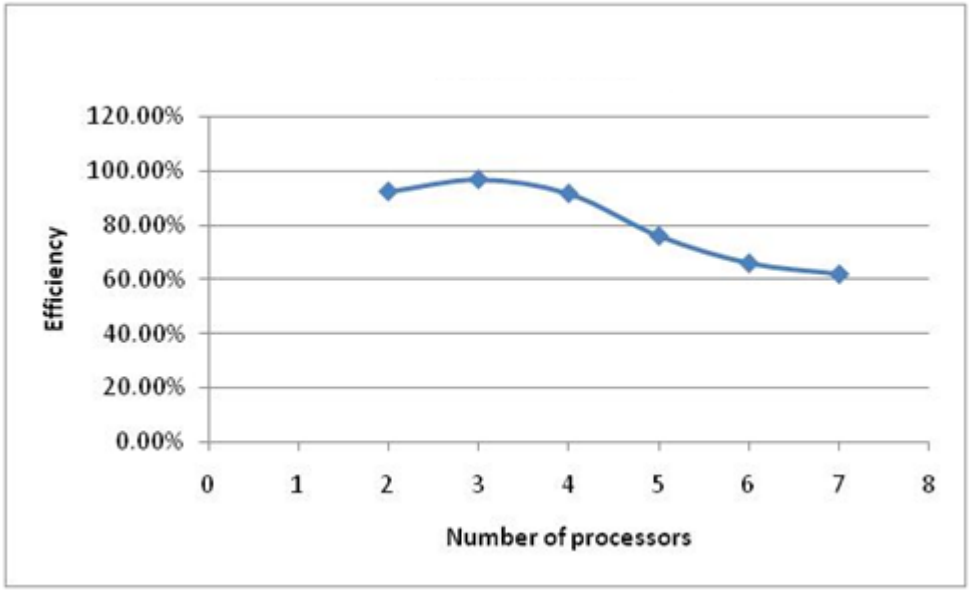



**Figure 5.36 - Maximum Efficiency when METIS was used on Test Case 5**

This decreasing trend in the efficiency values will eventually become too low for any advantage to be gained from the usage of the parallel system. Adding more processors only seem to increase the costs of running the simulations in parallel compared to the gain obtainable and METIS does not overcome the diminishing returns problems. This result suggests that this technique does not favour scalability.

The parallel simulation time performance ratios from using METIS are displayed in Table 5.29.

**Table 5.29 - Performance ratios over actual simulation times from using METIS on Test Case 5**

| Number of processors ↓ | Parallel SimTime[1] Performance ratio from using the METIS partitioning | | | | |
|---|---|---|---|---|---|
| No. of sub-domains → | P[2] sub-domains | 15 sub-domains | 25 sub-domains | 60 sub-domains | 100 sub-domains |
| **2** | 2.55 | 2.51 | 2.46 | 2.20 | 1.94 |
| **3** | 4.00 | 3.40 | 3.29 | 3.31 | 2.75 |
| **4** | 5.05 | 3.82 | 3.90 | 3.58 | 3.15 |
| **5** | 5.25 | 5.13 | 4.49 | 4.05 | 3.79 |
| **6** | 5.40 | 5.48 | 5.37 | 5.28 | 3.91 |
| **7** | 4.72 | 6.00 | 5.37 | 5.29 | 4.82 |

A maximum performance ratio with respect to the simulation time of 6.00 is not sufficiently fast to provide advice to incident commanders for live evacuations, yet is still a useful tool to employ in the investigations of evacuation scenarios for planning and possibly interactive training purposes.

Figure 5.37 demonstrates the maximum and minimum values of the speedups obtained when both METIS and the Cyclic Potential Route Map method (CPRM) were used. The graph also includes the ideal speedup achievable.

[1] **SimTime:** Simulation time (See Section 2.3.4)
[2] p: number of processors used (e.g 2, 3, …, 7)

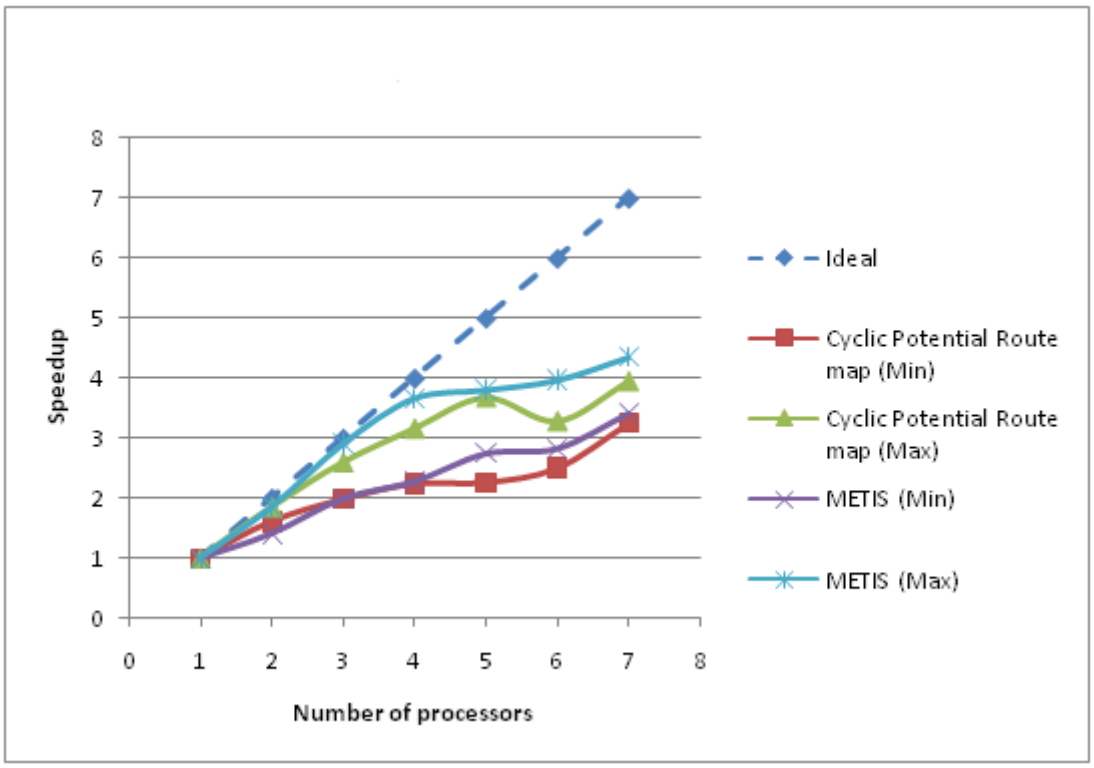



**Figure 5.37 - Comparisons of the CPRM method and METIS on Test Case 5**

When both methods were used, the results obtained are very good when up to four processors were used. Both curves tend to follow the linear trend that the ideal speedup suggests. Usually if the speedup is nearly linear, that is if it is nearly equal to the number of processors used, then according to Graham [149], the parallel system is said to be worthwhile. As the number of processors was increased, the speedup values obtained tend to decline which would indicate that the system is not scalable enough. This prompts concern as there is no guarantee that increasing the number of processors would produce better results.

Comparing METIS with the Cyclic Potential Route Map (CPRM) partitioning method, both methods seem to produce the same level of speedup values, with those from METIS being slightly better. Both methods seem to produce linear speedup up to the use of four processors and then their speedup values start to plateau after that point.

5.1.6.1.2 Test 2 – METIS partitioning

'Test Case 6 - 500m x 25m with one exit at one end and populated by 25,000 identical occupants' was partitioned by METIS. The average predicted simulation time to evacuate a population of 25,000 from this geometry is 833.5s The domain was partitioned into varying number of sub-domains, ranging from the number of available processors used to up to a hundred sub-domains.

The speedup values obtained are tabulated in Table 5.30.

**Table 5.30 - Results from using METIS partitioning on Test Case 6**

| Number of sub-domains ↓ | Speedup | | |
|---|---|---|---|
| Number of processors → | 2 processors | 3 processors | 4 processors |
| **p**[1] | 1.62 | 2.26 | 2.83 |
| **5** | 1.93 | 2.48 | 3.04 |
| **10** | 2.12 | 2.80 | 3.62 |
| **15** | 2.15 | 3.06 | 3.95 |
| **25** | 2.15 | 3.24 | 3.95 |
| **50** | 2.12 | 3.06 | 3.90 |
| **100** | 1.76 | 2.17 | 2.67 |

The speedup values obtained are very promising. When both two and three processors were used, super-linear speedup values were achieved and when four processors were used, a speedup value of 3.95 was obtained which is very close to the ideal speedup of 4. Since Test Case 6 produced very good results for the Cyclic method (i.e producing super-linear speedup no matter the number of processors used), the results obtained when METIS was used can only match those obtained previously and was not expected to be better than the results obtained previously.

[1] p: number of processors used (e.g 2, 3, …, 7)

Efficiency values are displayed in Table 5.31:

**Table 5.31 - Efficiency values when METIS was used on Test Case 6**

| Number of sub-domains ↓ | Efficiency | | |
|---|---|---|---|
| Number of processors → | 2 processors | 3 processors | 4 processors |
| p[1] | 81.07% | 75.18% | 70.69% |
| 5 | 96.50% | 82.54% | 75.98% |
| 10 | 105.93% | 93.50% | 90.40% |
| 15 | 107.33% | 102.06% | 98.75% |
| 25 | 107.54% | 108.01% | 98.68% |
| 50 | 105.79% | 101.92% | 97.44% |
| 100 | 87.81% | 72.47% | 66.74% |

From the efficiency values obtained, an efficiency of more than 100% implies that the processors were optimally used and the results are better than expected. Adding to the advantages of increased processor power and memory to run the algorithm, each processor's cache can now hold the data associated to its sub-domain without requiring the usage of more memory which would have been the case when only one processor was used.

To facilitate a direct comparisons to the previous methods used, Figure 5.38 illustrates the different speedup values obtained from using the different partitioning techniques attempted so far.

[1] p: number of processors used (e.g 2, 3, …, 7)

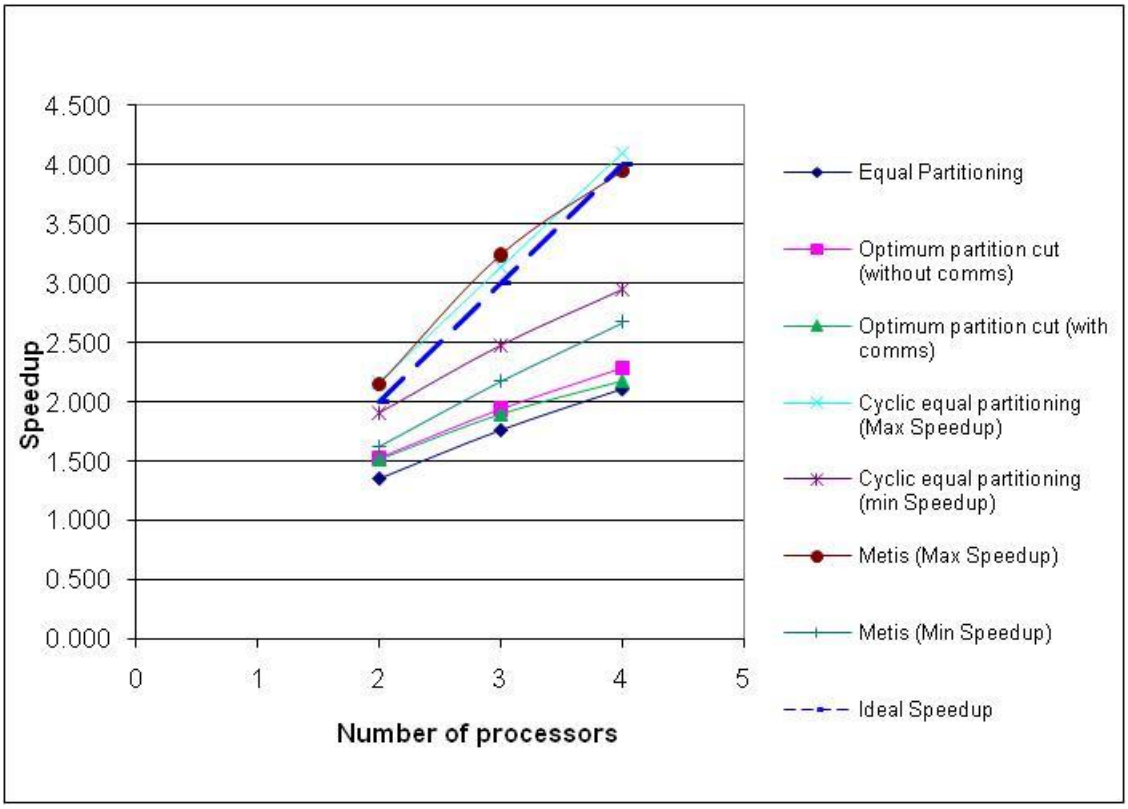



**Figure 5.38 - Comparisons of the different partitioning methods (including METIS) used on Test Case 6**

From Figure 5.38, METIS produces satisfactory results and implies that METIS is as productive as the cyclic method attempted before. The minimum speedup values obtained from using METIS are quite far from the ideal speedup values which suggest that choosing the right number of sub-domains is crucial in obtaining good speedup values. Looking at the minimum speedup values obtained from using the Cyclic Equal Partitioning method, they are closer to the ideal speedup values than those minimum values obtained from using METIS.

#### 5.1.6.1.3 Test 3 – METIS partitioning

'Test Case 8 - Replica of Beijing High Street randomly populated by 50,000 occupants with nine exits in total' was partitioned by METIS and different numbers of sub-domains were tested to find the best results. Up to eight processors were used for this test. The speedup values obtained are shown in Table 5.32:

**Table 5.32 - Speedup values from using different number of sub-domains in METIS on Test Case 8**

| Number of processors ↓ | Speedup values from using the METIS partitioning | | | | |
|---|---|---|---|---|---|
| **Number of processors →** | **p[1] sub-domains** | **15 sub-domains** | **25 sub-domains** | **60 sub-domains** | **100 sub-domains** |
| **2** | 1.647 | 1.834 | 1.785 | 1.608 | 1.515 |
| **3** | 2.837 | 2.222 | 1.798 | 2.148 | 1.952 |
| **4** | 2.837 | 2.510 | 2.053 | 2.401 | 2.105 |
| **5** | 4.353 | 3.152 | 3.928 | 2.583 | 2.384 |
| **6** | 5.092 | 4.162 | 2.863 | 2.932 | 2.520 |
| **7** | 4.800 | 4.295 | 3.832 | 2.887 | 2.314 |
| **8** | 5.223 | 4.083 | 3.777 | 3.658 | 2.647 |

METIS seems to produce better results when the geometry is being partitioned into NumProcs (number of processors) sub-domains. Here again as expected, when the domain is being divided into 100 sub-domains, the worst speedup values occur. The obvious reason is because the communication costs increase accordingly with the number of sub-domains. The choice in the optimum number of sub-domains to partition the problem is crucial to the success of the partitioning technique. From the difference in speedup values when different number of sub-domains was used, it can be seen that a sub-optimal number of sub-domains can be detrimental to the end results. Also, a set number of sub-domains cannot be found to implement for any number of processors used. For example partitioning the problem into p (Number of processors) sub-domains seems to give good results in most cases, but when two processors were used, this partitioning formulation did not produce the best results.

However, an interesting occurrence is that using six processors was more advantageous than when seven processors were used. Another interesting observation is the identical speedup values

[1] p: number of processors used (e.g 2, 3, …, 7)

of 2.837 when both three and four processors were used. This represents a waste of processor power as an extra processor should have increased the speedup accordingly and not be ineffective or indeed a hindrance to the parallel system. Using eight processors barely improved the speedup to 5.223 from the 5.092 obtained when six processors were used.

Table 5.33 gives a good indication on the performance of the parallel system by the efficiency values obtained.

**Table 5.33 - Efficiency values when METIS was applied to Test Case 8**

| **Number of processors ↓** | **Speedup values from using the METIS partitioning** | | | | |
|---|---|---|---|---|---|
| **Number of processors →** | **p sub-domains** | **15 sub-domains** | **25 sub-domains** | **60 sub-domains** | **100 sub-domains** |
| **2** | 82.33% | 91.70% | 89.23% | 80.41% | 75.74% |
| **3** | 94.57% | 74.07% | 59.95% | 71.60% | 65.07% |
| **4** | 70.93% | 62.76% | 51.34% | 60.02% | 52.62% |
| **5** | 87.05% | 63.04% | 78.55% | 51.66% | 47.68% |
| **6** | 84.87% | 69.36% | 47.71% | 48.87% | 41.99% |
| **7** | 68.57% | 61.35% | 54.74% | 41.25% | 33.05% |
| **8** | 65.29% | 51.03% | 47.21% | 45.72% | 33.09% |

When three processors were used, efficiency values ranging from 94.57% to 59.95% was achieved depending on the partitioning formulation. Again here, it can be seen that the choice of the right partitioning strategy can drastically affect the outcome of the parallelisation benefits. The maximum efficiency values obtained for the varying number of processors ranged from 94.57% to 65.29%, the former implying only about 5% of the processor usage was lost in parallel costs and idleness and the latter suggesting a performance decline with about 65% of each processor's capability being usefully applied in the running of the simulation and the rest being spent on costs and idleness.

The parallel simulation time performance ratios from using METIS with different number of sub-domains are displayed in Table 5.34.

**Table 5.34 - Performance ratios over actual simulation times from using different number of sub-domains in METIS on Test Case 8**

| **Number of processors ↓** | **Parallel SimTime[1] Performance ratio from using the METIS partitioning** | | | | |
|---|---|---|---|---|---|
| **Number of processors →** | **p[2] sub-domains** | **15 sub-domains** | **25 sub-domains** | **60 sub-domains** | **100 sub-domains** |
| **2** | 3.81 | 4.24 | 4.13 | 3.72 | 3.51 |
| **3** | 6.57 | 5.14 | 4.16 | 4.97 | 4.52 |
| **4** | 6.57 | 5.81 | 4.75 | 5.56 | 4.87 |
| **5** | 10.07 | 7.29 | 9.09 | 5.98 | 5.52 |
| **6** | 11.78 | 9.63 | 6.62 | 6.79 | 5.83 |
| **7** | 12.09 | 9.45 | 8.74 | 8.46 | 6.13 |
| **8** | 3.81 | 4.24 | 4.13 | 3.72 | 3.51 |

From Table 5.34, an encouraging maximum parallel performance ratio of 12.09 was achieved over the simulation time. This suggests that the use of a PC cluster consisting of 10 to 20 processors may provide sufficient computational performance to enable the use of advanced evacuation simulation for: live incident advice systems, reactive emergency signage systems, as well as investigations into types of evacuation scenarios for planning and training purposes.

Figure 5.39 demonstrates the speedup trend when both METIS and the Cyclic Potential Route Map method were used on Test Case 8.

[1] **SimTime:** Simulation time (See Section 2.3.4)
[2] p: number of processors used (e.g 2, 3, …, 7)

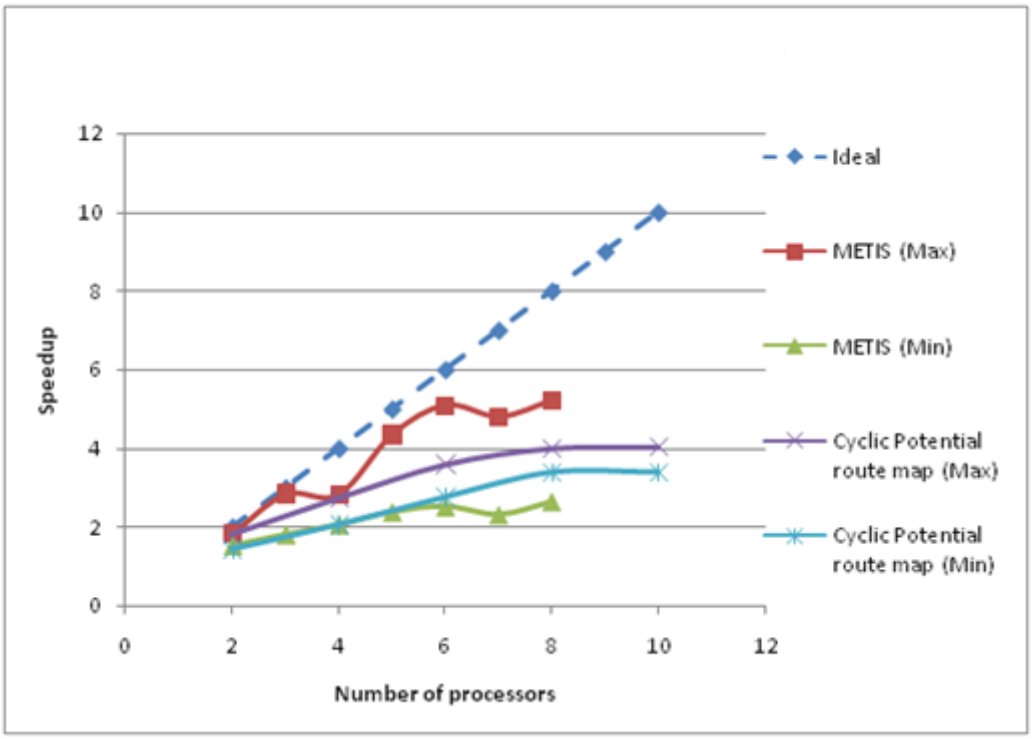



**Figure 5.39 - Comparison in speedup values from using the CPRM method and METIS on Test Case 8**

From the graph, it can be seen that METIS was more successful in partitioning Test Case 8 whenever the correct number of sub-domains were used (shown by the red line 'METIS (Max)'). The minimum speedup values from METIS are the lowest out of the two methods. The trend from the 'METIS (Max)' graph suggests a positive inclination, but it is also showing some oscillation and does not imply a smooth increasing trend like the ones obtained when the Cyclic Potential Route Map was used. This may imply an uncertainty on the reliability of the method.

## 5.2 *Concluding remarks*

Failing to obtain a scalable system is concerning as none of the methods attempted so far are proving to be the best one to adopt. The reason is the variations in the geometries tested lead to different partitioning methods being favoured. This suggests that it will be difficult to establish one approach that consistently improves results across different geometries. Another factor that makes the simulations so varied is the unique scenario of each case. The evacuation of people can

be unpredictable and can change during the course of the simulations due to other factors affecting the scenario (like sudden closure of exits, fire etc). These factors are enough to make a partitioning method redundant at certain times. For this type of model, a dynamic partitioning technique could resolve all these issues. By making the system dynamic in nature, the load imbalance and idleness problems would be resolved as the simulations will redistribute the sub-domains to idle processors as and when sudden changes occur.

# 6 DYNAMIC REPARTITIONING

From the previous attempts to find the optimal partitioning technique, no definite method was found to cater for the varied layouts and scenarios of each evacuation case. The different layouts can range from stadiums, whole cities or high rise buildings which consequently have different evacuation dynamics.

Another aspect of the simulation that a static partitioning struggles with is the unexpected phenomenon that can affect a simulation, like sudden closures of exits and the spread of fire amongst numerous incidents that can occur during an evacuation.

The main problem that the initial studies have shown is that a load imbalance affects the speed of the parallel system more than increased communication costs.

Dynamic repartitioning can potentially resolve all those issues brought on by varying layouts, scenarios, or the evacuation dynamics. By having dynamic repartitioning capabilities whereby load imbalance and processor activity are monitored, the problems associated with the static partitioning should be resolved. The geometry will be repartitioned to ensure each processor will be allocated the same amount of work to prevent load imbalance.

As demonstrated, the static partitioning technique cannot efficiently handle the general nature of an evacuation let alone the potential unexpected nature of an evacuation with possible evolving scenarios. Once the initial allocated regions are created, they can only stay on that processor. Hence, if a region becomes empty quicker or becomes unusable due to a sudden closure of that region, then the processors handling those domains will become load imbalanced. A dynamic repartitioning technique can check those scenarios and re-assign those idle processors to regions with population activities.

A possible disadvantage that can be associated with the dynamic repartitioning technique is increased data migration. In addition to having the usual communications occurring with the movement of the population; whenever there is a repartitioning the geometry and population data must be communicated between the processors. Consequently there will be an increase in

communication costs. However, from the static tests performed in the previous chapter, it was found that having a load imbalance and idleness is more detrimental than having an increase in communication costs. Hence, the dynamic repartitioning may be a more viable option.

## 6.1 Dynamic repartitioning Strategy

With the dynamic repartitioning strategy; at certain points in the simulation, the algorithm assesses the current situation to decide whether there is a need to repartition the domain. There are two main methods considered for this repartitioning technique:

1. Completely formulate a new set of partitions every time dynamic repartitioning is required (see Figure 6.1)

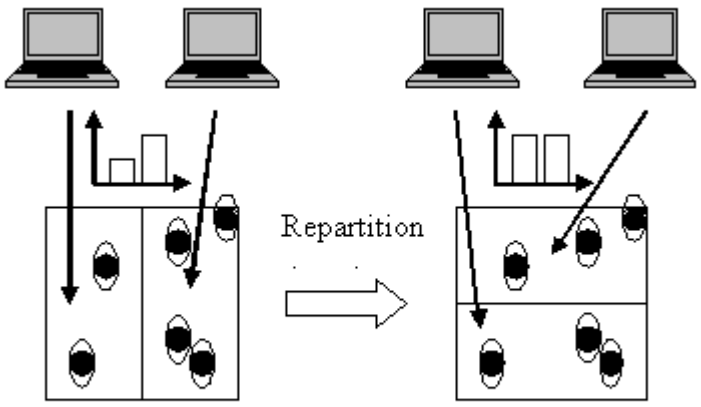



**Figure 6.1 – Illustration of dynamic repartition**

2. Keep the original partitions and swaps them around to promote load balancing, similarly to what Minyard et al. [138] considered (see Figure 6.2).

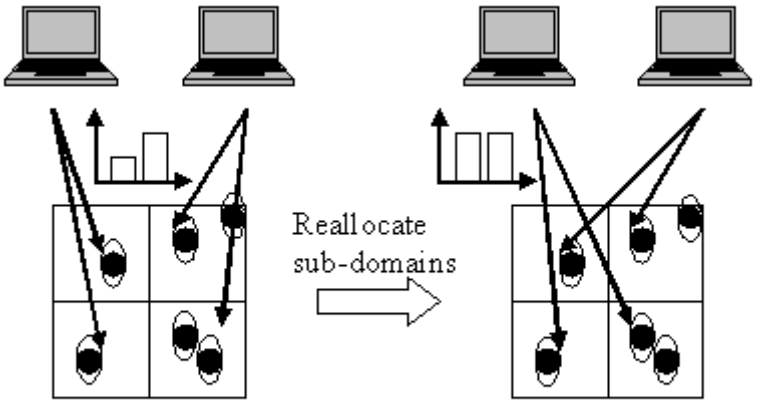



**Figure 6.2 – Illustration of dynamic reallocation**

The first method seems attractive in the respect that a completely new set of partitions is created every time there is a change in conditions and this can imply fewer partitions to be created and hence reducing the communication costs. However, this method will require significant data migration and to determine the new partitions, as well as find and assign halo nodes. Data migration consists of moving both the nodes and the population. Though the nodes exist on all the processors, they have dynamic properties associated to them and these properties get updated only on the processor they reside in. A new partitioning strategy may mean that the set of nodes which resided on one processor may now be residing on different processors, making the tracking and communication of these nodes more complicated. The same problems will arise in the communication of the population. Population residing on one processor will now be communicated to various processors (one to many) instead of just being transferred to one other (one to one) processor. In short, this method favours the one to many communicating strategy which involves more data tracking as well as the computations required to work out the new halo nodes and assigning them to the relevant processors.

The second method consists of keeping the original partitions created at the start of the simulation and only swaps them around based on the idleness and load of each processor. By keeping the original partitions, the cost of calculating new halos at each re-allocation is avoided as well as the complication of communicating data across a variety of processors. By having a “one to on” communication system, each processor has to communicate a set of data to only one other

processor rather than having to break this set and communicate parts of it to different processors. In short, a dynamic reallocation occurs rather than a dynamic repartitioning.

### *6.1.1 Suggested dynamic reallocation algorithm*

At regular intervals in the simulation, each processor checks for any load imbalance in accordance with a set criterion such as population load, workload or cost function. If a load imbalance to a set tolerance is found to occur, then the decision to reallocate the problem is made. Each processor communicates its local load across and runs a suitable algorithm to redistribute the load as equally as possible. At this point each processor will have different sub-domains allocated to it. The process of exchanging their local loads (both local nodes and population) is performed which results in each processor having a new set of partitions, thus enabling the processors to have roughly the same load.

#### 6.1.1.1 *Creation of Subparts*

Some modifications to the source codes had to be implemented so as to facilitate the dynamic reallocation strategy. One of the main additions is the creation of an object class called Subparts. This object has various attributes necessary for the ease of reallocating the problem. Each Subpart contains the local nodes and people in a partitioned sub-domain. So the geometry is partitioned into Subparts each of which is linked to a unique processor which can have several Subparts. Each processor is aware of the load it is currently servicing by requesting the information from its local Subparts.

#### 6.1.1.2 *Added multi-halo regions to permit dynamic reallocation*

Multi-halo nodes get created so that each processor does not have more than two processors to a particular part of a region as explained in 4.2.4.1.1. The highlighted section of the domain in Figure 6.3 demonstrates where a set of multi-halo have to be created. The nodes in the highlighted section on the region catered for by processor Red would have been in the halo regions of processors Blue and Cyan. Since in EXODUS it is the node that determines the

movement of the individuals, this particular node exists on three processors and is a problem for the movement algorithm. Hence, a set of multi-halo nodes (depicted by the thin light green strips) (Section 4.2.4.1.1) was created along the regions shared by several processors to remediate this problem.

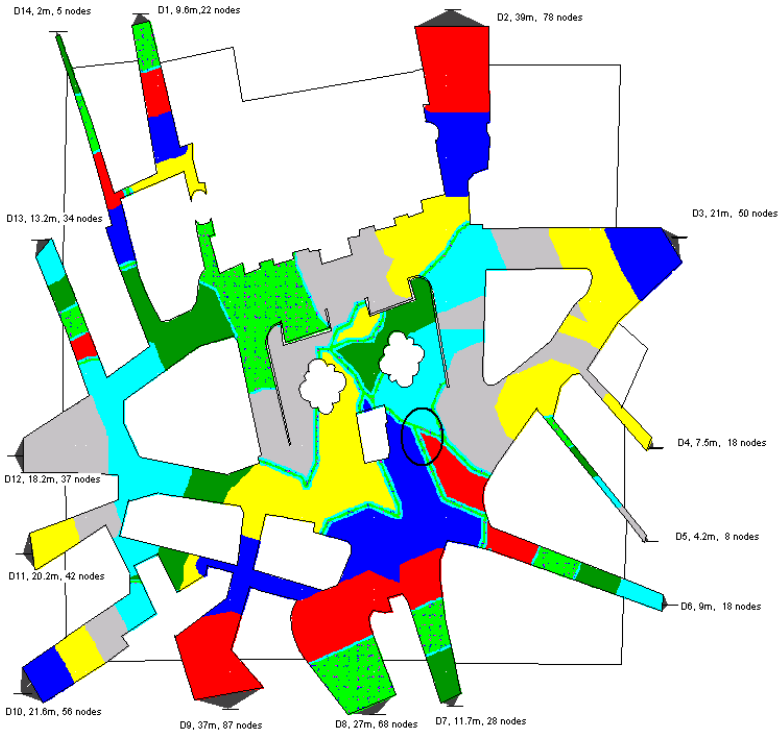
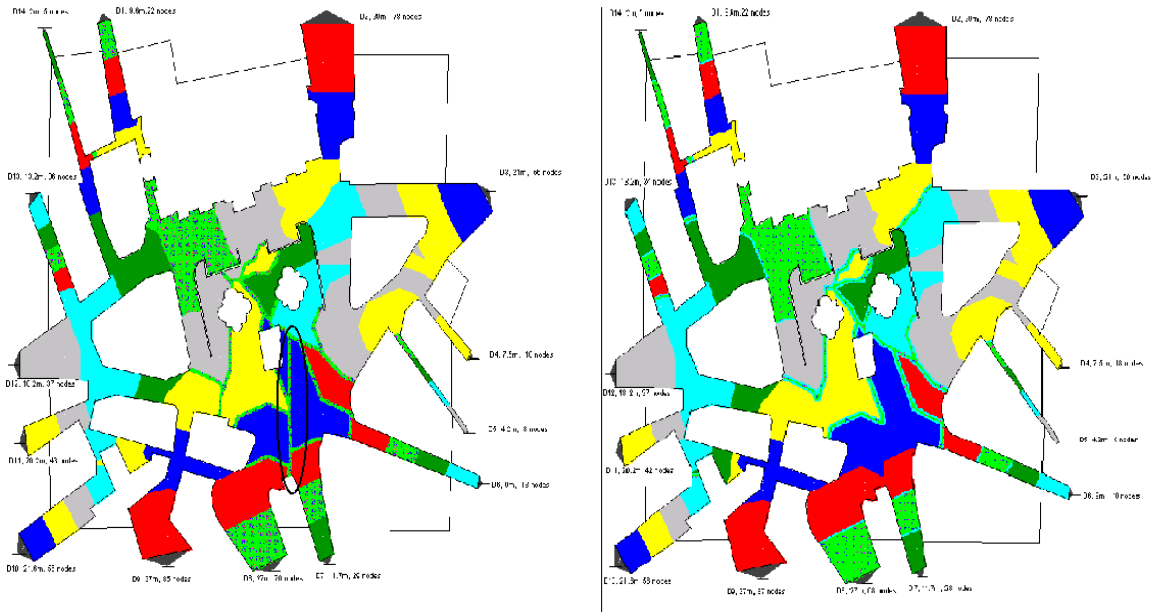



**Figure 6.3 - An example of Multi-halo nodes on the Trafalgar Square replica**

Multi-halo nodes are calculated before the simulation starts and in static mode they remain unchanged. These multi-halo nodes were calculated in the assumption that all Subparts are allocated to predetermined processors and will not be reallocated to different ones throughout the simulation. In the above example, there are some neighbouring Subparts which are serviced by the same processor and hence there is no need to have either halo nodes or multi-halo ones. However, in the dynamic reallocation strategy, these Subparts may be allocated to different processors and then there will arise the need to have halo nodes. It is computationally expensive to employ the halo finding algorithm every time a reallocation is required.

Due to the lack of time to fully explore and solve this problem, a simple solution was devised to counter this problem. At the start of the simulation, each Subpart was treated as belonging to a unique processor and hence all the possible halos and multi-halo nodes were calculated at the start of the simulation. This strategy eliminates the need to recalculate halo and multi-halo nodes each time a reallocation occurs, but also creates more halo and multi-halo nodes than is necessary at times (when two neighbouring processors are serviced by the same processors); see Figure 6.4. The highlighted oval selection in Figure 6.4(a) shows multi-halo nodes being created, separating two Subparts which appear as one when in static mode Figure 6.4(b). These two Subparts might need to be reallocated to different processors in the dynamic mode and hence multi-halo nodes would have been necessary.



**(a)** Dynamic: Addition of multi-halo nodes **(b)** Static

**Figure 6.4 - Addition of multi-halo nodes at some boundaries**

Ideally, a complete set of new partitions based on the current simulation would have been desirable. However, this cannot be explored at present due to the time constraints. Hence the adopted strategy is the simple solution explained above; which eliminates the high computation costs associated in recalculating the halo nodes at every reallocation thus offsetting the advantages gained by dynamically repartitioning a problem. However, increased communication

cost is expected due to the higher number of multi-halo nodes created. From the previous tests performed, it was found that the communication cost is negligible compared to having a load imbalance, so this method is adopted for the moment until another partitioning strategy can be explored.

### *6.1.2 Method to redistribute the load*

The prominent method to schedule and redistribute a load is the LPT (Longest processing time) algorithm and uses the concept of sorting the load in descending order [151, 152]. The LPT algorithm sorts a collection of jobs according to the processing time to run those jobs then assign them to the different processors.

#### 6.1.2.1 *LPT (Longest processing time) algorithm used for this research*

For this problem, the main factor affecting the parallel system is the load imbalance. Hence the load of each processor, instead of the actual processing time that the common LPT algorithm employs, is used in the current formulation. The collection of Subparts is sorted in descending order according to their current workload. Each processor gets allocated the Subparts at the top of the list in such a fashion that the processor with the minimum load gets assigned the Subpart with the largest load. This process is done until all the Subparts have been allocated.

A prototype program was written to test this algorithm. An example of the functionality of this program is now presented.

A set of random numbers is generated and the numbers have to be ideally allocated to six processors so that they all have roughly the same total. These numbers were partitioned by both the LPT algorithm [151, 152] and JOSTLE [146] in order to compare the final partitioned results.

92 numbers were randomly generated and they were:

| | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 90 | 50 | 87 | 80 | 20 | 81 | 7 | 96 | 51 | 39 | 81 | 98 | 2 | 73 | 56 | 44 | 3 | 97 | 34 |
| 55 | 99 | 17 | 85 | 69 | 39 | 46 | 95 | 20 | 52 | 31 | 14 | 74 | 20 | 3 | 97 | 97 | 89 | 40 |
| 91 | 91 | 33 | 96 | 61 | 64 | 93 | 34 | 40 | 32 | 66 | 54 | 65 | 89 | 23 | 23 | 65 | 81 | |
| 85 | 13 | 31 | 80 | 14 | 81 | 91 | 2 | 48 | 81 | 5 | 99 | 16 | 18 | 50 | 4 | 3 | 63 | |
| 60 | 61 | 41 | 90 | 35 | 78 | 80 | 4 | 33 | 8 | 98 | 94 | 82 | 14 | 79 | 7 | 21 | 83 | |

The LPT algorithm scheduled and assigned this load to six processors as displayed in Table 6.1:

**Table 6.1 - Scheduling based on the LPT algorithm**

| | **Total load** | **Load ratio**<br>**Ideal ratio = 16.67%** | **Number of items** | **Numbers** |
|---|---|---|---|---|
| **P1** | 831 | 16.6734% | 15 | 99 94 91 85 81 78 65 55 52 40 33 31 16 7 4 |
| **P2** | 831 | 16.6734% | 15 | 99 93 91 89 80 73 69 54 51 40 35 23 17 14 3 |
| **P3** | 831 | 16.6734% | 16 | 98 96 90 83 81 80 64 60 50 39 33 21 20 8 5 3 |
| **P4** | 830 | 16.6533% | 15 | 98 95 91 82 81 80 65 56 50 44 32 20 20 14 2 |
| **P5** | 831 | 16.6734% | 16 | 97 97 89 87 81 74 66 61 48 39 34 31 14 7 4 2 |
| **P6** | 830 | 16.6533% | 15 | 97 96 90 85 81 79 63 61 46 41 34 23 18 13 3 |

JOSTLE was used to allocate the same list of numbers to six processors and the results are illustrated in Table 6.2:

**Table 6.2 - Scheduling performed by JOSTLE**

| | Total load | Load ratio<br>Ideal ratio = 16.67% | Number of items | Numbers |
|---|---|---|---|---|
| **P1** | 854 | 17.1348% | 15 | 80 80 80 79 73 63 61 50 50 44 41 40 40 39 34 |
| **P2** | 829 | 16.6332% | 9 | 99 99 96 95 93 91 91 83 82 |
| **P3** | 852 | 17.0947% | 28 | 87 78 74 65 61 56 55 52 51 48 39 33 23 20 17 16 14 14 13 7 7 5 4 3 3 3 2 2 |
| **P4** | 853 | 17.1148% | 22 | 89 85 66 65 64 60 54 46 35 34 33 32 31 31 23 21 20 20 18 14 8 4 |
| **P5** | 816 | 16.3724% | 9 | 98 97 97 97 91 89 85 81 81 |
| **P6** | 780 | 15.6501% | 9 | 98 96 94 90 90 81 81 81 69 |

As can be seen from the final schedule, both methods enabled the processors to have roughly the required 16.6667% of the total load. However, comparing the two algorithms, the LPT algorithm provides a better load balance in terms of both the ratio of the total load as well as in the number of items on each processor. JOSTLE's distribution of the total load was balanced enough but its distribution of the number of items was not as balanced as was possible and shown by the LPT algorithm. Thus the LPT algorithm was used to schedule the loads of the Subparts to ensure that each processor ends up with roughly the same load as well as a similar number of Subparts as possible.

### 6.1.3 Data migration

Since the LPT is run on all the processors, each processor is aware of which parts of the domains belong to which processors; so no communications of this data is necessary at this stage. Consequently when reallocation happens, the population in the local domains of each processor needs to be communicated to the processor which now runs this domain (one to one communication).

Data migration is efficiently performed by the pack routine that MPI [37] provides. The same 'pack and send' 'and 'unpack and receive' routines used in the communication of people at boundaries during the simulation are used when reallocation occurs. For each processor, the main components of the data migration routine are explained below:

- Pack, send and delete the people who were in myOldDomain.
- Release myOldDomain and associate myNewDomain to belonging in my local domain.
- Update the halo nodes with respect to myNewDomain.
- Receive, unpack and create the people who are now in myNewDomain.

### 6.1.4 Load reallocation at the start of the simulation with no dynamic reallocation

At the start of the simulation when the partitioning algorithm is called, there is no guarantee that this first partition creates a load balanced system. Inevitably, applying a dynamic reallocation technique generates extra costs (added communications and reshuffling costs). An approach to avoid a dynamic reallocation is to perform a population load balancing at the start and then letting the system run in static mode.

This strategy was tested by the following test cases:

**Test Case 9 - The Trafalgar Square replica was used, with seven of the fourteen exits closed and with a population of 60,000.**

This scenario of having half the number of exits closed is presented to show how the static partitioning technique is compelled to stay with the original partitions that were created before the simulation. The average predicted simulation time to evacuate a population of 60,000 from this

geometry is 5,015s. Hence, some processors which are simulating regions with closed exits may eventually make the system load imbalanced. A reallocation based on population load should make the system load balanced at the start of the simulation, but does not guarantee that same factor throughout the simulation.

Eight processors were used to test this idea and the Cyclic Potential Route Map (CPRM) technique was used with varying number of sub-parts at each exit. The speedup values obtained are displayed in Table 6.3:

**Table 6.3 - Speedup and efficiency values obtained on Test Case 9; when an initial reallocation is done at the start of the simulation and the system remains static**

| Number of sub-parts at each exit | Total number of sub-domains | Static Parallel system | | Initial load balancing, followed by static system | |
|---|---|---|---|---|---|
| | | Speedup | Efficiency | Speedup | Efficiency |
| **1** | 14 | 1.839 | 23.00% | 2.026 | 25.33% |
| **2** | 28 | 2.194 | 27.43% | 2.315 | 28.94% |
| **4** | 55 | 2.779 | 34.74% | 2.921 | 36.51% |
| **6** | 78 | 3.376 | 42.20% | 3.199 | 39.99% |

There is only a marginal increase in the speedup values obtained when using this strategy. The results obtained are expected, since the initial reallocation is only based on population load rather than the current situation of the scenario (half of the exits are closed), the algorithm may allocate some processors to the regions containing the closed exits. Since the system is statically partitioned, these processors are forced to simulating the same domains and hence load imbalance eventually occurs. A maximum speedup value of 3.376 was achieved when no initial population load balance was performed and a value of 3.199 when there was a population load balancing at the start of the simulation. Both values do not suggest a good performance from the parallel system of eight processors.

This test was performed with a scenario of having seven of the fourteen exits closed. Before discarding this idea, another set of tests with a different scenario was applied. The following Test Case 10 uses the same geometry used in Test Case 9, but applying a scenario of having all its exits open, but the population is now centred in the domain.

**Test Case 10 - Trafalgar Square replica populated with 60,000 people located towards the centre of the geometry**

The average predicted simulation time to evacuate a population of 60,000 from this geometry is 986.67s. Both the CPRM technique and METIS (with a varying number of sub-domains) were used as partitioning techniques, on various number of processors (2, 4, 6 and 8). The runs were performed in static mode (Full Static) then with initial load balance reallocation at the start of a static mode (Part Static). The results obtained are given in Table 6.4 for speedup values and in Table 6.5 for the efficiency values:

**Table 6.4 - Speedup values obtained on Test Case 10; when an initial reallocation is done at the start of the simulation and the system remains static**

| | CPRM | | | | METIS | | |
|---|---|---|---|---|---|---|---|
| **No. of procs** | **Num. of sub-parts per exit** | **Total Num. of sub-domains** | **Speedup Full Static** | **Speedup Part Static** | **Num. of sub-domains** | **Speedup Full Static** | **Speedup Part Static** |
| **2** | **1** | **14** | 1.405 | 1.492 | **15** | 1.722 | 1.580 |
| | **2** | **28** | 1.695 | 1.736 | **30** | 1.683 | 1.564 |
| | **4** | **55** | 1.740 | 1.780 | **60** | 1.814 | 1.770 |
| | **6** | **78** | 1.819 | 1.603 | **90** | 1.644 | 1.607 |
| **4** | **1** | **14** | 2.096 | 2.317 | **15** | 2.567 | 2.929 |
| | **2** | **28** | 2.280 | 2.563 | **30** | 2.977 | 2.648 |
| | **4** | **55** | 2.511 | 2.688 | **60** | 2.632 | 2.830 |
| | **6** | **78** | 2.448 | 2.734 | **90** | 2.724 | 2.753 |
| **6** | **1** | **14** | 2.566 | 3.102 | **15** | 3.145 | 3.318 |
| | **2** | **28** | 2.942 | 3.506 | **30** | 3.817 | 3.449 |
| | **4** | **55** | 3.127 | 2.999 | **60** | 3.801 | 3.361 |
| | **6** | **78** | 3.310 | 3.457 | **90** | 3.282 | 3.563 |
| **8** | **1** | **14** | 2.570 | 3.345 | **15** | 3.856 | 3.933 |
| | **2** | **28** | 2.812 | 3.424 | **30** | 3.144 | 3.745 |
| | **4** | **55** | 2.618 | 3.416 | **60** | 3.990 | 4.152 |
| | **6** | **78** | 2.970 | 3.389 | **90** | 3.503 | 3.148 |

**Table 6.5 - Efficiency values obtained on Test Case 10; when an initial repartitioning is done at the start of the simulation and the system remains static**

| | CPRM | | | | METIS | | |
|---|---|---|---|---|---|---|---|
| **No. of procs** | **No. subparts per exit** | **Total No. subparts** | **Efficiency Full Static** | **Efficiency Part Static** | **No. of subparts** | **Efficiency Full Static** | **Efficiency Part Static** |
| **2** | **1** | **14** | 70.25% | 74.60% | **15** | 86.10% | 79.00% |
| | **2** | **28** | 84.75% | 86.80% | **30** | 84.15% | 78.20% |
| | **4** | **55** | 87.00% | 89.00% | **60** | 90.70% | 88.50% |
| | **6** | **78** | 90.95% | 80.15% | **90** | 82.20% | 80.35% |
| **4** | **1** | **14** | 52.40% | 57.93% | **15** | 64.18% | 73.23% |
| | **2** | **28** | 57.00% | 64.08% | **30** | 74.43% | 66.20% |
| | **4** | **55** | 62.78% | 67.20% | **60** | 65.80% | 70.75% |
| | **6** | **78** | 61.20% | 68.35% | **90** | 68.10% | 68.83% |
| **6** | **1** | **14** | 42.77% | 51.70% | **15** | 52.42% | 55.30% |
| | **2** | **28** | 49.03% | 58.43% | **30** | 63.62% | 57.48% |
| | **4** | **55** | 52.12% | 49.98% | **60** | 63.35% | 56.02% |
| | **6** | **78** | 55.17% | 57.62% | **90** | 54.70% | 59.38% |
| **8** | **1** | **14** | 32.13% | 41.81% | **15** | 48.20% | 49.16% |
| | **2** | **28** | 35.15% | 42.80% | **30** | 39.30% | 46.81% |
| | **4** | **55** | 32.73% | 42.70% | **60** | 49.88% | 51.90% |
| | **6** | **78** | 37.13% | 42.36% | **90** | 43.79% | 39.35% |

The results obtained after doing a reallocation of the original partitions do not show any improvements. In certain cases, this modification worsened the results and at other cases only minimally improved the static runs. This suggests that all the processors are working optimally at the start of the simulation on the original partitions created. The problem seems to be at a later stage of the simulation, when there is a possibility of load imbalance and processor idleness. Hence, a dynamic reallocation strategy is desired to monitor the load imbalance whenever it arises.

### *6.1.5 When to reallocate*

Each second in EXODUS is broken down by 1/12 of a second and this is termed as a tick; please refer to Section 3.2.1. At every odd tick, the movement of the population is determined and at each even tick movement actually occurs [30].

Ideally, reallocation should occur sufficiently often such that the gain from reallocation is always greater than letting the system run in load imbalance. This was an unknown area in the early stage of testing the dynamic reallocation techniques. Hence, some tests were required to get an idea of how expensive a reallocation is compared to the gain achieved from having an improved load balance.

To start testing the dynamic reallocation codes, a starting criterion of a 75% population load imbalance was used to warrant a reallocation. If the processors were found to be 75% less than the most overloaded processors, a load balancing is performed. The next question was at which time interval the requirement is tested.

Some tests were run by varying the time interval when the dynamic reallocation algorithm is called.

#### 6.1.5.1 *Time interval of 1s (12 ticks) and 10s (120 ticks)*

‘Test Case 9 - The Trafalgar Square replica was used, with seven of the fourteen exits closed and with a population of 60,000.’ was used to test at which time intervals the dynamic reallocation algorithm should be called. Eight processors were used and the Cyclic Potential Route Map (CPRM) method was used with each exit generating from two to six sub-partitions. The speedup

obtained, as well as the number of times the reallocation of the geometry happened was recorded for two different time scenarios; at each second interval and at each 10 second intervals. Since the reallocation costs is still an unknown quantity, those two values were chosen to see how influential the number of times the reallocation algorithm is called.

The results obtained are tabulated in Table 6.6.

**Table 6.6 - Speedup and number of times reallocation happened when dynamic reallocation algorithm called at 1s and then at 10s intervals**

| | **Speedup** | |
|---|---|---|
| | **1s (12 ticks)** | **10s (120 ticks)** |
| **1 sub-part/exit = 14 sub-domains in total** | 1.713 | 1.747 |
| **No. of times reallocation happened** | 2 | 2 |
| **2 sub-parts/exit = 28 sub-domains in total** | 1.789 | 1.811 |
| **No. of times reallocation happened** | 8 | 4 |
| **4 sub-parts/exit = 55 sub-domains in total** | 2.593 | 2.671 |
| **No. of times reallocation happened** | 34 | 9 |
| **6 sub-parts/exit = 78 sub-domains in total** | 3.803 | 3.962 |
| **No. of times reallocation happened** | 40 | 11 |

Using eight processors, the maximum speedup achieved is a disappointing 3.962. Running this test case without the dynamic reallocation capabilities produced a maximum speedup of 3.376 displayed in Table 6.3. However, these tests were run in order to get a measure of the reallocation costs from the frequency the algorithm is called. The interesting fact is that the speedup values are very similar in both cases. This similarity in the speedup values suggests that the calls to the algorithm are not computationally expensive. With 10 times more calls to the reallocation algorithm when the 1s interval was made, similar speedup values were achieved. Another interesting discovery is the number of times the geometry was repartitioned when both cases were run. As expected, calling the algorithm to determine a load imbalance at every second compared to calling it at every 10s induced more reallocations. When the geometry was

partitioned into six sub-parts at each exit, calling the dynamic algorithm every second found that on 40 occasions the system was less than 75% population load balanced and was load imbalanced on 11 occasions when called every 10s. By having similar speedup values suggest that the reallocation costs are not drastically expensive and the number of times reallocation can happen is no longer a factor to be minimised or avoided.

#### 6.1.5.2 *Reallocation algorithm called every 30 ticks together with a 75% population load imbalance criteria*

From the previous test, calling the reallocation algorithm at either every second or every 10 seconds produced similar results. The purpose of the dynamic reallocation algorithm is to improve the results obtained from using a static partitioning strategy. Some more tests were devised to verify that there is an improvement in using the dynamic reallocation capabilities. The time interval at which to call the dynamic reallocation algorithm to verify that the system is load balanced is still an unknown quantity. So this time, the dynamic reallocation algorithm is being called at every 30 ticks which is equivalent to 2.5s. Every 2.5s the software assesses the load balancing of the problem and if there is a load imbalance of less than 75%, then the problem is being reallocated to resolve this problem. Test Case 10 (Section 6.1.4) was used for testing the dynamic reallocation algorithm. The average predicted simulation time to evacuate a population of 60,000 from this geometry is 986.67s.

The uneven distribution of the population is to show how the dynamic algorithm can adapt to having those scenarios and the static partitioning strategy is forced to handle the same domains assigned at the start of the simulation. The number of processors used ranged from two to eight. The Cyclic Potential Route Map (CPRM) method was used as the partitioning strategy with varying number of sub-parts.

Legends for the terms used in the table:

P: Number of processors used

N: Number of sub-parts

The results are displayed in Table 6.7:

**Table 6.7 - Comparison in the speedup values from using both the static and dynamic reallocation techniques on Test Case 10**

| P | | | CPRM | | | | | |
|---|---|---|---|---|---|---|---|---|
| | N/exit | Total No. sub-domains | Speedup | | % change | Efficiency | | No. reallocations |
| | | | Static | Dynamic | | Static | Dynamic | |
| 2 | 1 | 14 | 1.405 | 1.797 | 27.90% | 70.25% | 89.85% | 28 |
| | 2 | 28 | 1.695 | 1.871 | 10.38% | 84.75% | 93.55% | 28 |
| | 4 | 56 | 1.740 | 1.802 | 3.56% | 87.00% | 90.10% | 29 |
| | 6 | 83 | 1.819 | 1.779 | -2.20% | 90.95% | 88.95% | 17 |
| 4 | 1 | 15 | 2.096 | 3.085 | 47.19% | 52.40% | 77.13% | 27 |
| | 2 | 29 | 2.280 | 3.244 | 42.28% | 57.00% | 81.10% | 29 |
| | 4 | 57 | 2.511 | 2.942 | 17.16% | 62.78% | 73.55% | 47 |
| | 6 | 84 | 2.448 | 2.940 | 20.10% | 61.20% | 73.50% | 38 |
| 6 | 1 | 14 | 2.566 | 2.866 | 11.69% | 42.77% | 47.77% | 34 |
| | 2 | 28 | 2.942 | 3.483 | 18.39% | 49.03% | 58.05% | 28 |
| | 4 | 57 | 3.127 | 3.434 | 9.82% | 52.12% | 57.23% | 47 |
| | 6 | 84 | 3.310 | 3.341 | 0.94% | 55.17% | 55.68% | 42 |
| 8 | 1 | 15 | 2.570 | 2.945 | 14.59% | 32.13% | 36.81% | 13 |
| | 2 | 29 | 2.812 | 3.284 | 16.79% | 35.15% | 41.05% | 21 |
| | 4 | 57 | 2.618 | 3.409 | 30.21% | 32.73% | 42.61% | 33 |
| | 6 | 84 | 2.970 | 3.286 | 10.64% | 37.13% | 41.08% | 20 |

The goal to improve the results from using a static partitioning strategy to a dynamic one has been achieved. In most cases, the speedup values obtained from using the dynamic method are better than the ones achieved by the static method. However, the speedup values are still too low when the number of processors increases. When two processors were used, a maximum speedup value of 1.871 was obtained with the dynamic reallocation strategy, which is very efficient. Even

the maximum speedup value of 3.244 is considered efficient when four processors were used. However, speedup values of 3.483 and 3.284 when six and eight processors were used are not promising at all. The only positive note is that all the values obtained from the dynamic reallocation method improved its corresponding static strategy values.

Even the number of reallocations was not expensive enough to degrade the results achieved by the dynamic algorithm. Again, this proves that a load imbalanced problem is worse than a load balanced one together with the added costs necessary to reallocate the partitions. Consider the results obtained when four processors were used with two sub-parts at each exit; a speedup value of 2.280 was obtained when the problem did not have any dynamic reallocation capabilities and a value of 3.244 when it did. This problem was found to be less than 75% load imbalanced on 29 occasions and a full reallocation happened every time. Even with this added costs of reallocating the whole problem 29 times, a significant improvement to the speedup value was achieved. Hence, the dynamic reallocation code has improved the software's running time compared to the static partitioning strategy.

The total number of sub-domains represented in the table shows different values depending on the number of processors and of sub-domains used. As can be seen in the cases of when two and eight processors being used, the total numbers of sub-domains created are different because the 'thin part removal' algorithm (Section 4.3.3) has removed some sub-domains due to their sizes.

#### 6.1.5.3 *Pop-load vs Work-load*

The time to run the software may be significantly different depending on the level of congestion, even if there is a low population load. The dynamics of the movement algorithm changes considerably when there is congestion involved. A simulation containing a population load with no congestion will run quicker than a scenario dealing with the same population size but with congestion involved. Thus the simulation is more dependent on the work load the program is going through, rather than the population size it is trying to move. Since the dynamic reallocation algorithm tested in the previous section uses the population load (pop-load) criteria to decide a reallocation, it may have been using the wrong criteria to determine the load imbalance. It is assumed that it will involve more operations to simulate the evacuation of a population when there is a packed situation and lesser operations to simulate that same population in a sparsely

populated area. To confirm these assumptions, some tests were run to determine how much work-load is associated to a same population but in two different scenarios. The first scenario (Sparse scenario) comprises of putting a population of 20,000 occupants in a geometry measuring 50m by 500m with a free-flowing exit along the length of the enclosure so that it is sparsely populated and no major congestion occurred during the evacuation. The second scenario (Packed scenario) involves using the same population and enclosure but with only a 10m exit so that there is congestion whilst the evacuation is simulated.

Figure 6.5 illustrates the results tested.

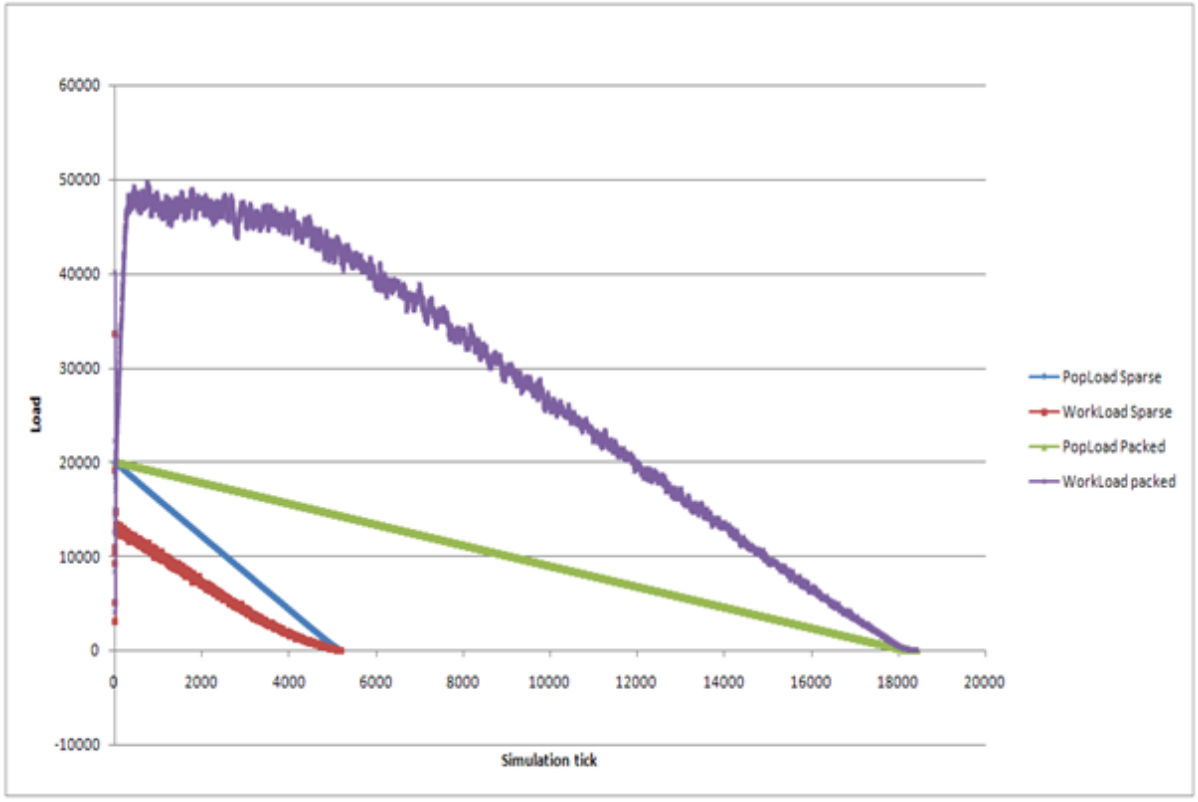



**Figure 6.5 - Difference between Population Load and Work-load**

Legend:

The "Load" from the y-axis in Figure 6.5 represents two different metrics based on whether the PopLoad or the WorkLoad was used, as explained below:

**PopLoad (Blue and Green graphs):** Number of people

**WorkLoad (Red and Purple graphs):** Number of CPU operations

The graph in Figure 6.5 shows that the population load has gradual linear decrease as the simulation progresses and does not reflect any other aspects (like congestions) that may be affecting the simulation. However, by considering the number of operations (work-load) involved in moving the population, in the case of the packed scenario the distribution obtained does not follow a decreasing linear distribution but rather peaked at the point of expected congestion. However, in the sparse scenario the work-load curve follows the decreasing trend observed from the pop-load case as it does not involve extra operations needed to simulate congestion. Both

workload distributions from the two different scenarios reflect the condition of the current situation in a simulation rather that the population load distributions which only reflect the remaining population at that point. These work-load lines show that the work associated to move a population is not linearly dependent of that population and has other operations involved at certain times of the simulation. Comparing the two work-load lines, which represent the operations involved to move the same population, shows different behaviours based on the current situation of the simulation. When there was a packed geometry, the work-load is much more elevated than the workload incurred when the geometry was sparsely populated.

Hence the work-load criterion, rather than using the population load, is preferred in estimating the current load balance of the parallel system and has been implemented in the dynamic reallocation algorithm.

#### 6.1.5.4 *Cost function to determine when to reallocate*

No definite tests can ascertain at which regular intervals or at which ratio of load imbalance warrant a reallocation. These factors do not consider the current situation of the parallel system, such as network latency or memory usage of the processors. So dynamically reallocating a simulation at a busy network period might not bring the same advantages that reallocating at a low load network time will provide. Without a dynamic factor to assess the current parallel system, basing the decision to reallocate the problem on a random value of 75% load imbalance at a certain time interval might be reallocating the problem at the wrong time.

Hence a dynamic factor has to be taken into account to decide whether to reallocate at any time in the simulation. A mathematical gain formula was considered to assess the performance of the parallel system for any repartitioning. Minyard et al. [138] also came up with a similar scheme to take into consideration the current situation of the parallel system. The idea is to use the latest data obtained when the last reallocation occurred as well as the time it took to shift a population load. This formula is called the gain formula and it comprises of the costs of reallocating a problem at the previous reallocation. These costs comprise of receiving and sending a certain population, the costs of updating the halos and the movement of the population. The gain formula is shown in Eq (8):

$$Gain = C_{Old} - (C_{New} + C_{Reallocation}) \quad \textbf{Eq (8)}$$

Legends:

$C_{New}$ : Costs of running new workload

$C_{Old}$ : Costs of running old workload

$C_{Reallocation}$ : Reallocation costs

If Gain > 0, then reallocate the problem.

The cost of running the new workload $(C_{New})$is found as follows:

- Each processor gets its projected new load after applying the LPT to its current workload.
- The costs of running this new workload is calculated by taking the slowest workload costs obtained at the previous movement operations.
- The minimum cost by the busiest processor is adopted.

The cost of running the old workload $(C_{Old})$ is found as follows:

- The cost of running the old workload is calculated in the same fashion as the method used to get the new workload cost.

The reallocation cost $(C_{Reallocation})$ is made up as follows:

- Costs of updating the halos.
- Costs of sending the old population.

- Costs of receiving the new population.
- The maximum cost obtained by the most overloaded processor is adopted.

A detailed description of the dynamic reallocation algorithm can be found in Appendix 9.7.

6.1.5.5 *Dynamic re-allocation algorithm*

The breakthrough in finding the Gain formula to dynamically determine when there is a load imbalance in the parallel system eventually led to the adopted Dynamic re-allocation algorithm displayed in Figure 6.6.

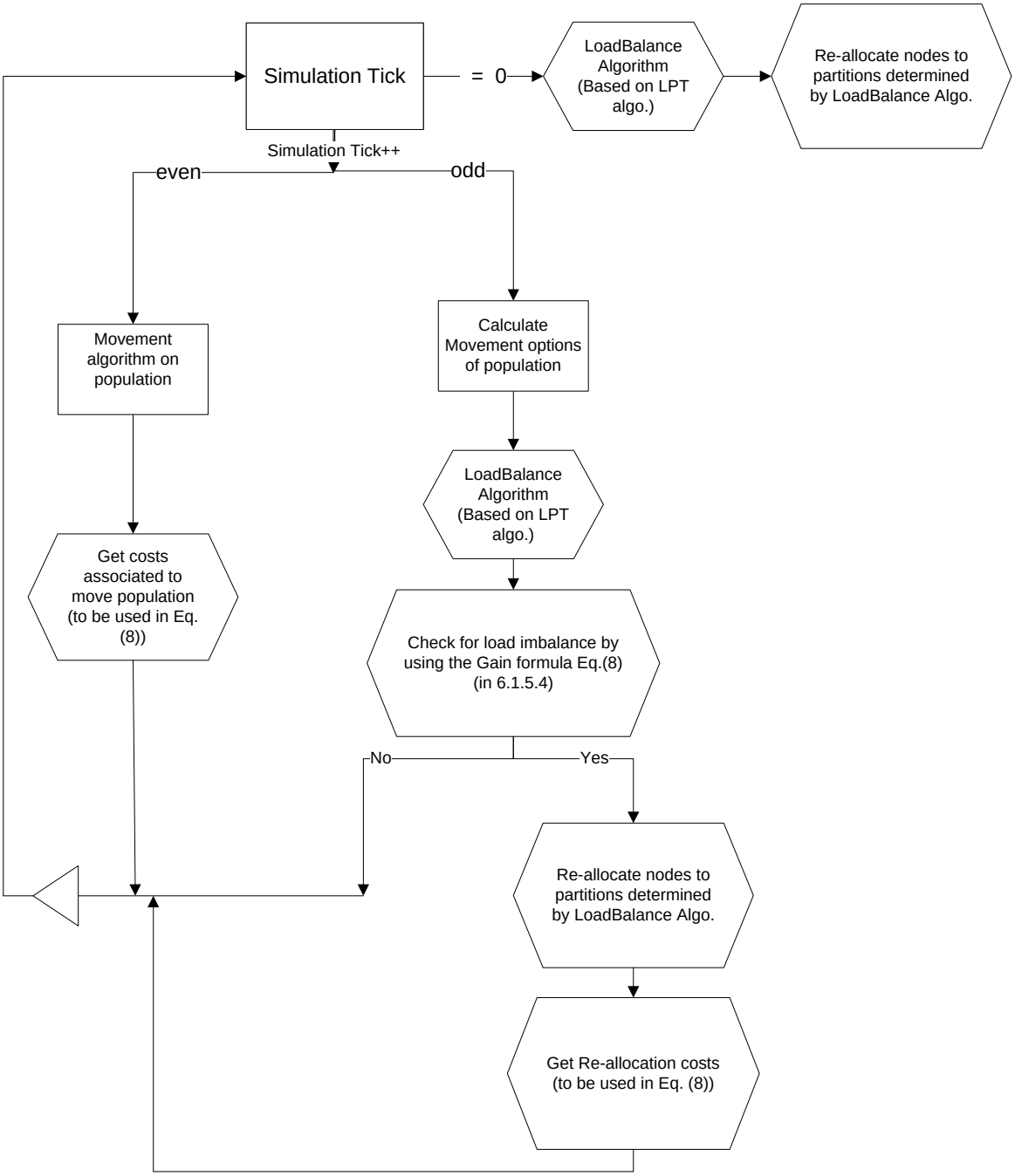



**Figure 6.6 - Dynamic re-allocation algorithm**

6.1.5.5.1 Testing the dynamic reallocation algorithm using the CPRM partitioning strategy

This new algorithm was tested on 'Test Case 10 - Trafalgar Square replica populated with 60,000 people located towards the centre of the geometry' as a direct comparison to the results in Table 6.7 which were obtained from the previous dynamic reallocation algorithm being called at every 30 ticks and reallocate the problem if there was a 75% or below load imbalance). As before, the Cyclic Potential Route Map partitioning strategy is applied with different number of sub-parts tested. Two to eight processors were used in the parallel system.

Legends for the terms used in the table:

**CPRM:** Cyclic Potential Route Map method

**P:** Number of processors used

**N:** Number of sub-domains

**S**: Number of sub-parts at each exit

**NR:** Number of times reallocation occurred

**GF:** Gain formula

The average predicted simulation time to evacuate a population of 60,000 from this geometry is 986.67s. The speedup values are displayed in Table 6.8 and the efficiency values in Table 6.9:

**Table 6.8 - Speedup values obtained when the latest formulation of the dynamic reallocation algorithm (including the gain function) was applied to Test Case 10.**

| P | CPRM | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | S per exit | Total N | Speedup | | | Speedup | | |
| | | | Static | Dynamic (30 ticks) | NR | Dynamic (GF) | NR | % change Dynamic (GF) v/s Static |
| 2 | 1 | 14 | 1.405 | 1.797 | 28 | 1.839 | 79 | 30.89% |
| | 2 | 28 | 1.695 | 1.871 | 28 | 1.800 | 79 | 6.19% |
| | 4 | 56 | 1.740 | 1.802 | 29 | 1.822 | 24 | 4.71% |
| | 6 | 83 | 1.819 | 1.779 | 17 | 1.788 | 24 | -1.70% |
| 4 | 1 | 15 | 2.096 | 3.085 | 27 | 3.031 | 14 | 44.61% |
| | 2 | 29 | 2.280 | 3.244 | 29 | 2.953 | 50 | 29.52% |
| | 4 | 57 | 2.511 | 2.942 | 47 | 3.021 | 51 | 20.31% |
| | 6 | 84 | 2.448 | 2.940 | 38 | 2.972 | 63 | 21.41% |
| 6 | 1 | 14 | 2.566 | 2.866 | 34 | 3.251 | 2 | 26.70% |
| | 2 | 28 | 2.942 | 3.483 | 28 | 3.546 | 3 | 20.53% |
| | 4 | 57 | 3.127 | 3.434 | 47 | 3.748 | 66 | 19.86% |
| | 6 | 84 | 3.310 | 3.341 | 42 | 3.715 | 100 | 12.24% |
| 8 | 1 | 15 | 2.570 | 2.945 | 13 | 3.320 | 2 | 29.18% |
| | 2 | 29 | 2.812 | 3.284 | 21 | 3.167 | 3 | 12.62% |
| | 4 | 57 | 2.618 | 3.409 | 33 | 3.337 | 9 | 27.46% |
| | 6 | 84 | 2.970 | 3.286 | 20 | 3.638 | 70 | 22.49% |

**Table 6.9 - Efficiency values obtained when the latest formulation of the dynamic reallocation algorithm (including the gain function) was applied to Test Case 10.**

| P | CPRM | | | | | | |
|---|---|---|---|---|---|---|---|
| | S per exit | Total N | Speedup | | | Speedup | |
| | | | Static | Dynamic (30 ticks) | NR | Dynamic (GF) | NR |
| 2 | 1 | 14 | 70.25% | 89.85% | 28 | 91.95% | 79 |
| | 2 | 28 | 84.75% | 93.55% | 28 | 90.00% | 79 |
| | 4 | 56 | 87.00% | 90.10% | 29 | 91.10% | 24 |
| | 6 | 83 | 90.95% | 88.95% | 17 | 89.40% | 24 |
| 4 | 1 | 15 | 52.40% | 77.13% | 27 | 75.78% | 14 |
| | 2 | 29 | 57.00% | 81.10% | 29 | 73.83% | 50 |
| | 4 | 57 | 62.78% | 73.55% | 47 | 75.53% | 51 |
| | 6 | 84 | 61.20% | 73.50% | 38 | 74.30% | 63 |
| 6 | 1 | 14 | 42.77% | 47.77% | 34 | 54.18% | 2 |
| | 2 | 28 | 49.03% | 58.05% | 28 | 59.10% | 3 |
| | 4 | 57 | 52.12% | 57.23% | 47 | 62.47% | 66 |
| | 6 | 84 | 55.17% | 55.68% | 42 | 61.92% | 100 |
| 8 | 1 | 15 | 32.13% | 36.81% | 13 | 41.50% | 2 |
| | 2 | 29 | 35.15% | 41.05% | 21 | 39.59% | 3 |
| | 4 | 57 | 32.73% | 42.61% | 33 | 41.71% | 9 |
| | 6 | 84 | 37.13% | 41.08% | 20 | 45.48% | 70 |

The goal of the dynamic reallocation was to improve the speedups obtained when a static partitioning strategy was used, especially as the number of processors is increasing. One of the main problems with the static partitioning method was that the system was not showing signs of scalability. From the results displayed in Table 6.8 and Table 6.9, most of the speedup values from the dynamic strategy were better than those obtained previously when the static method was used. So, there is definitely a success in improving the speedup values. Even with the high number of reallocation, which reached up to a 100, an improvement from 3.341 to 3.715 was achieved when 6 processors partitioned the problem into 84 sub-domains. But when the number of reallocations was low, no marked improvements were incurred. For example when eight processors were used to partition the problem into 29 sub-domains, the dynamic reallocation only happened three times and no improvements in the speedup values were achieved (3.284 and 3.167) and only slightly improved the results. These latest values again confirm that increased communication costs incurred by the dynamic reallocation are not a major parallelisation cost compared to letting the system become imbalanced.

The best speedup values obtained for each P are shown in Figure 6.7, which illustrates the comparison between the two partitioning strategies.

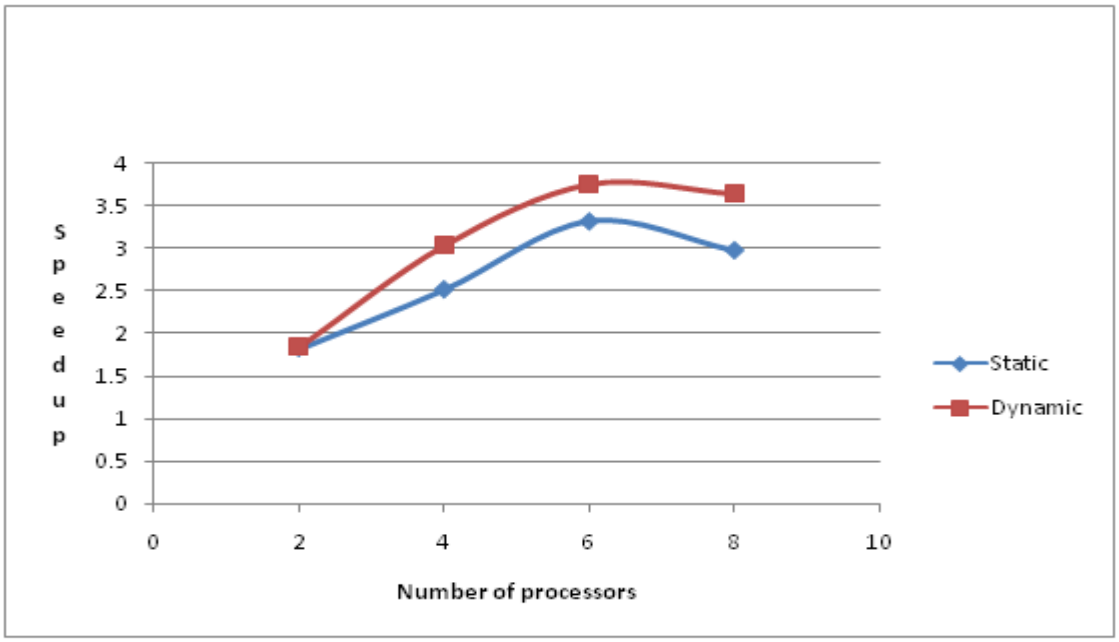



**Figure 6.7 - Comparison between the static and dynamic partitioning strategies when the CPRM was used**

Figure 6.7 clearly shows a definite improvement in the speedup values obtained when the dynamic reallocation strategy was applied. The only worrying factor is the scalability of the parallel system. The dynamic reallocation graph seems to decline when eight processors were used, suggesting no scope for scalability. However, due to the lack of processors in the cluster used in this research, this theory could not be tested further.

#### 6.1.5.5.2 Test on the dynamic reallocation algorithm using METIS partitioning strategy

Since the results from using the CPRM partitioning method did not indicate a scope for the scalability of the system, METIS was used in an attempt to surmount this problem. The number of sub-domains used by METIS to partition the problem was chosen so that they would be roughly equal to the ones used by the CPRM method.

Legends for the terms used in the table:

**CPRM:** Cyclic Potential Route Map method

**N:** Number of sub-domains

**NR:** Number of times reallocation occurred

**GF:** Gain formula

**P:** Number of processors used

The average predicted simulation time to evacuate a population of 60,000 from this geometry is 986.67s. The speedup values obtained are displayed in Table 6.10 and the efficiency values in Table 6.11.

**Table 6.10 - Speedup values obtained when METIS was applied to Test Case 10 using the finalised Dynamic reallocation algorithm (using the gain formula).**

| P | METIS | | | | | CPRM | | |
|---|---|---|---|---|---|---|---|---|
| | Total N | Speedup | | | % Change | Total N | Speedup | |
| | | Static | Dynamic (GF) | NR | | | Dynamic (GF) | NR |
| 2 | 15 | 1.722 | 1.840 | **37** | 6.85% | 14 | 1.839 | 79 |
| | 30 | 1.683 | 1.846 | **37** | 9.69% | 28 | 1.800 | 79 |
| | 60 | 1.814 | 1.790 | **22** | -1.32% | 56 | 1.822 | 24 |
| | 89 | 1.644 | 1.696 | **19** | 3.16% | 83 | 1.788 | 24 |
| 4 | 15 | 2.929 | 3.038 | **83** | 3.72% | 15 | 3.031 | 14 |
| | 30 | 2.648 | 3.255 | **73** | 22.92% | 29 | 2.953 | 50 |
| | 60 | 2.830 | 2.961 | **65** | 4.63% | 57 | 3.021 | 51 |
| | 89 | 2.753 | 2.722 | **65** | -1.13% | 84 | 2.972 | 63 |
| 6 | 15 | 3.145 | 3.931 | **60** | 24.99% | 14 | 3.251 | 2 |
| | 30 | 3.817 | 3.783 | **92** | -0.89% | 28 | 3.546 | 3 |
| | 60 | 3.801 | 3.732 | **105** | -1.82% | 57 | 3.748 | 66 |
| | 89 | 3.282 | 3.201 | **128** | -2.47% | 84 | 3.715 | 100 |
| 8 | 15 | 3.856 | 4.157 | **5** | 7.81% | 15 | 3.320 | 2 |
| | 30 | 3.144 | 4.081 | **64** | 29.80% | 29 | 3.167 | 3 |
| | 59 | 3.990 | 3.541 | **114** | -11.25% | 57 | 3.337 | 9 |
| | 89 | 3.503 | 3.097 | **133** | -11.59% | 84 | 3.638 | 70 |

**Table 6.11 - Efficiency values obtained when METIS was applied to Test Case 10 using the finalised Dynamic reallocation algorithm (using the gain formula).**

| P | METIS | | | | CPRM | | |
|---|---|---|---|---|---|---|---|
| | Total N | Efficiency | | | Total N | Efficiency | |
| | | Static | Dynamic (GF) | NR | | Dynamic (GF) | NR |
| 2 | 15 | 86.10% | 92.00% | **37** | 14 | 91.95% | 79 |
| | 30 | 84.15% | 92.30% | **37** | 28 | 90.00% | 79 |
| | 60 | 90.70% | 89.50% | **22** | 56 | 91.10% | 24 |
| | 89 | 82.20% | 84.80% | **19** | 83 | 89.40% | 24 |
| 4 | 15 | 73.23% | 75.95% | **83** | 15 | 75.78% | 14 |
| | 30 | 66.20% | 81.38% | **73** | 29 | 73.83% | 50 |
| | 60 | 70.75% | 74.03% | **65** | 57 | 75.53% | 51 |
| | 89 | 68.83% | 68.05% | **65** | 84 | 74.30% | 63 |
| 6 | 15 | 52.42% | 65.52% | **60** | 14 | 54.18% | 2 |
| | 30 | 63.62% | 63.05% | **92** | 28 | 59.10% | 3 |
| | 60 | 63.35% | 62.20% | **105** | 57 | 62.47% | 66 |
| | 89 | 54.70% | 53.35% | **128** | 84 | 61.92% | 100 |
| 8 | 15 | 48.20% | 51.96% | **5** | 15 | 41.50% | 2 |
| | 30 | 39.30% | 51.01% | **64** | 29 | 39.59% | 3 |
| | 59 | 49.88% | 44.26% | **114** | 57 | 41.71% | 9 |
| | 89 | 43.79% | 38.71% | **133** | 84 | 45.48% | 70 |

From the values obtained when METIS' partitioning strategy was used, a slight improvement was seen when the dynamic reallocation algorithm was applied over those obtained by the static partitioning method. Again, the dynamic reallocation algorithm is found to be equally good or better than its static counterpart; the implementation of the dynamic reallocation routine will

adapt to any geometry or scenario that each simulation can offer. As long as the speedup values obtained by the dynamic reallocation technique are not much worse than the static method was achieving, this new algorithm is the appropriate to implement.

The number of times the problem was reallocated does not seem to be a major cost which could deteriorate the speedup values. As can be seen from Table 6.10, any improvement in the speedup values does not seem to be dependent on the number of times the problem was reallocated (NR). When eight processors were used to partition the problem into 15 sub-domains, a speedup value of 4.157 was improved from 3.856 and only five reallocations were necessary. However, 73 reallocations were needed to improve the speedup from 2.648 to 3.255 when four processors were used on the geometry partitioned into 30 sub-domains.

The parallel simulation time performance ratios from using both the CPRM and METIS partitioning technique are displayed in Table 6.12.

**Table 6.12 - Performance ratios over actual simulation times when METIS and CPRM were applied to Test Case 10 using the finalised Dynamic reallocation algorithm (using the gain formula).**

| P | METIS | | CPRM |
|---|---|---|---|
| | Total N | Parallel SimTime[1] Performance ratio | Parallel SimTime Performance ratio |
| 2 | 15 | 2.73 | 2.73 |
| | 30 | 2.74 | 2.67 |
| | 60 | 2.65 | 2.70 |
| | 89 | 2.51 | 2.65 |
| 4 | 15 | 4.50 | 4.49 |
| | 30 | 4.83 | 4.38 |
| | 60 | 4.39 | 4.48 |
| | 89 | 4.03 | 4.40 |
| 6 | 15 | 5.83 | 4.82 |
| | 30 | 5.61 | 5.26 |
| | 60 | 5.53 | 5.55 |
| | 89 | 4.74 | 5.51 |
| 8 | 15 | 6.16 | 4.94 |
| | 30 | 6.05 | 4.69 |
| | 59 | 5.25 | 4.95 |
| | 89 | 4.59 | 5.39 |

From Table 6.12, neither method could provide a high enough parallel performance ratio over the simulation time to warrant the parallel implementation to be used in a live evacuation. However,

[1] **SimTime:** Simulation time (See Section 2.3.4)

the performance enhancements achieved contributes in investigating types of evacuation scenarios for planning and training purposes.

Figure 6.8 shows the difference in the speedup values obtained from using both the static partitioning and dynamic reallocation strategies partitioned by both the CPRM method and the METIS software.

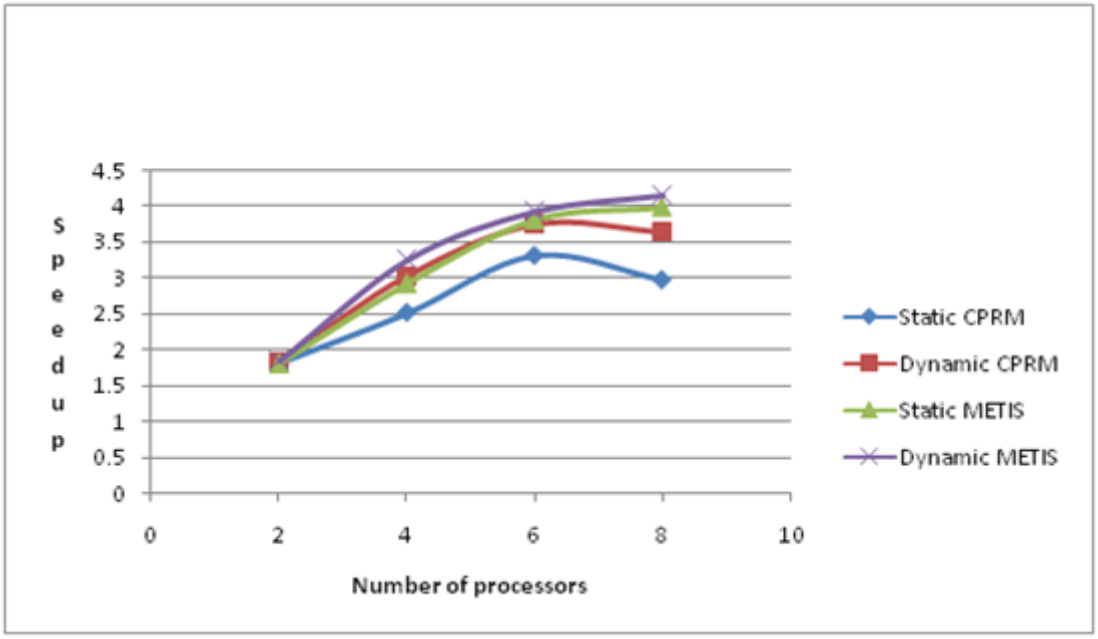



**Figure 6.8 - Comparison between the static and dynamic partitioning strategies from using both CPRM and METIS**

From the graph above, METIS is providing the better speedup values than the CPRM method. The dynamic reallocation strategy is consistently better than using the static method. The drawback from using the CPRM method is that the scalability factor is not looking promising, which is a considerable disadvantage to the expansion of the problem size and/or the addition of more processor resources. However, using METIS with a dynamic reallocation strategy is showing a better sign of scalability compared to using the CPRM method. As the number of processors increases so does the speedup which suggests that the system is scalable.

Another unknown quantity is the optimum number of sub-domains to partition the geometry into. As can be seen from the results obtained so far, partitioning the geometry into the "wrong" number of sub-domains can lead to poor speedups from either too much communication costs or from poor load balance. From the tests performed in this chapter, an analysis can be performed to

determine the number of required sub-domains. Figure 6.9 illustrates which number of sub-domains gave the best speedup values as the number of processors was increased.

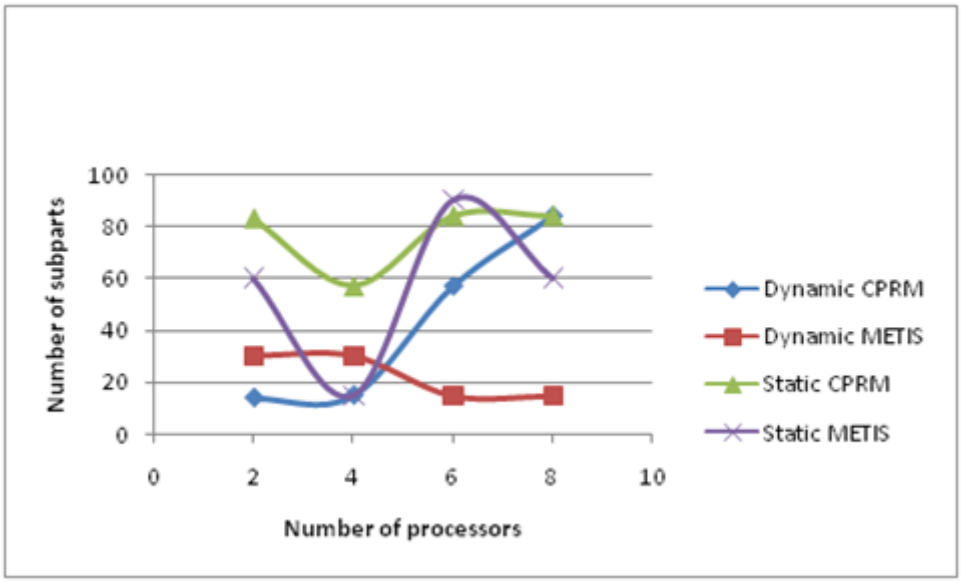



**Figure 6.9 - Number of sub-domains which gave the best speedup values**

An interesting pattern in the required number of sub-domains to use was discovered when the dynamic reallocation strategy was applied. When METIS is being used, as the number of processors was increased, decreasing the number of sub-domains used provided the best speedup values. Alternately, in the case when the CPRM method was used in the dynamic reallocation algorithm, the opposite phenomenon was observed. The lower the number of processors used, the lower the number of sub-domains to be used. Increasing the number of processors suggests increasing the number of sub-domains accordingly should provide the best results. However, these findings do not apply if the static partitioning method is being used.

#### 6.1.5.5.3 Test to show how both the dynamic reallocation algorithm and static partitioning method cope with a particular scenario

Due to the dynamic nature of an evacuation scenario where the simulation can incorporate any new developments as the evacuation is proceeding, having a static partitioning strategy will struggle to cope with any changes to the original configurations. On the other hand, a dynamic

reallocation method can assess any changes in the dynamics of the simulation and reallocate the problem accordingly to promote load balance.

An example was used so that the evacuation scenario does not favour the static partitioning method and how the dynamic reallocation algorithm can surmount those problems.

**Test Case 11 - Rectangular geometry measuring 500 x 700 nodes with 24 exits, 8 operational and 16 closed.**

This geometry is randomly populated by 80,000 occupants. Eight processors were used in the parallel system for all cases shown in this section.

Figure 6.10 illustrates the geometric configuration of Test Case 11.

Legends:

: Exit is not operational

: Serviceable exit

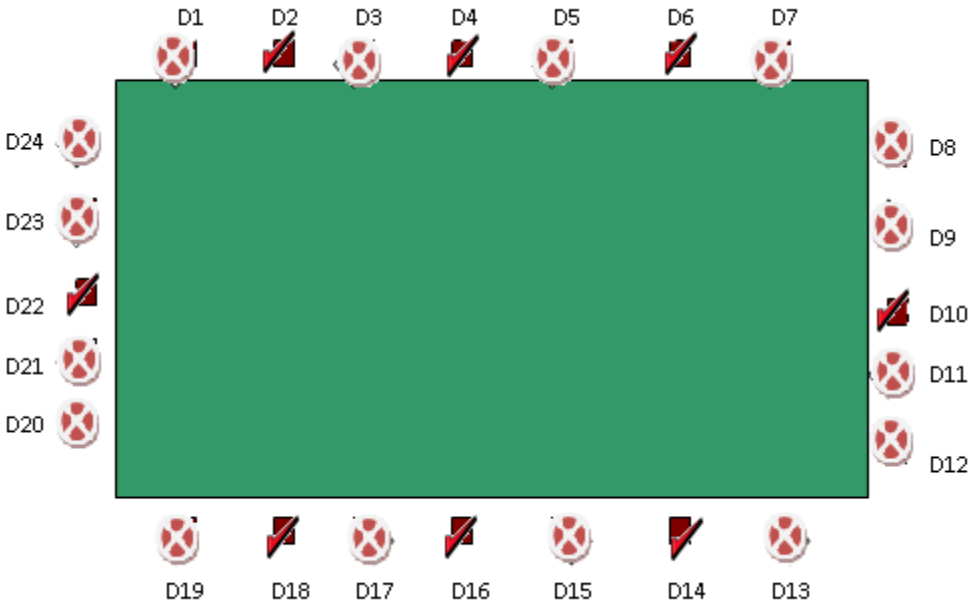



**Figure 6.10 - Illustration of Test Case 11**

With this configuration set up, Figures Figure 6.11 Figure 6.13 show the initial population distribution at 0s then at 30s and at 70s.

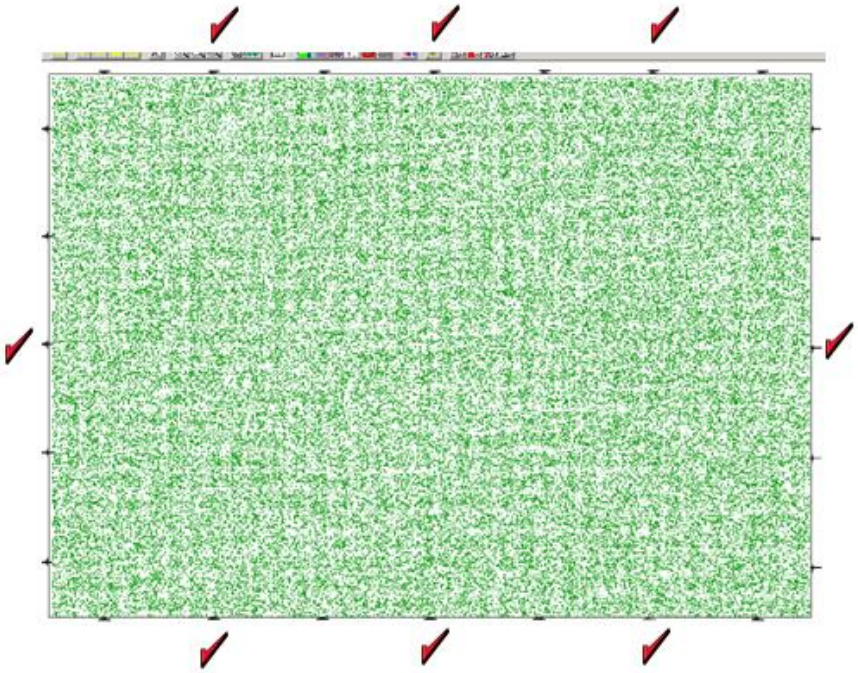


**Figure 6.11 - Population distribution at 0s**

At 30s in the simulation, a pattern can be seen in the route the population is heading; which is to the serviceable exits.

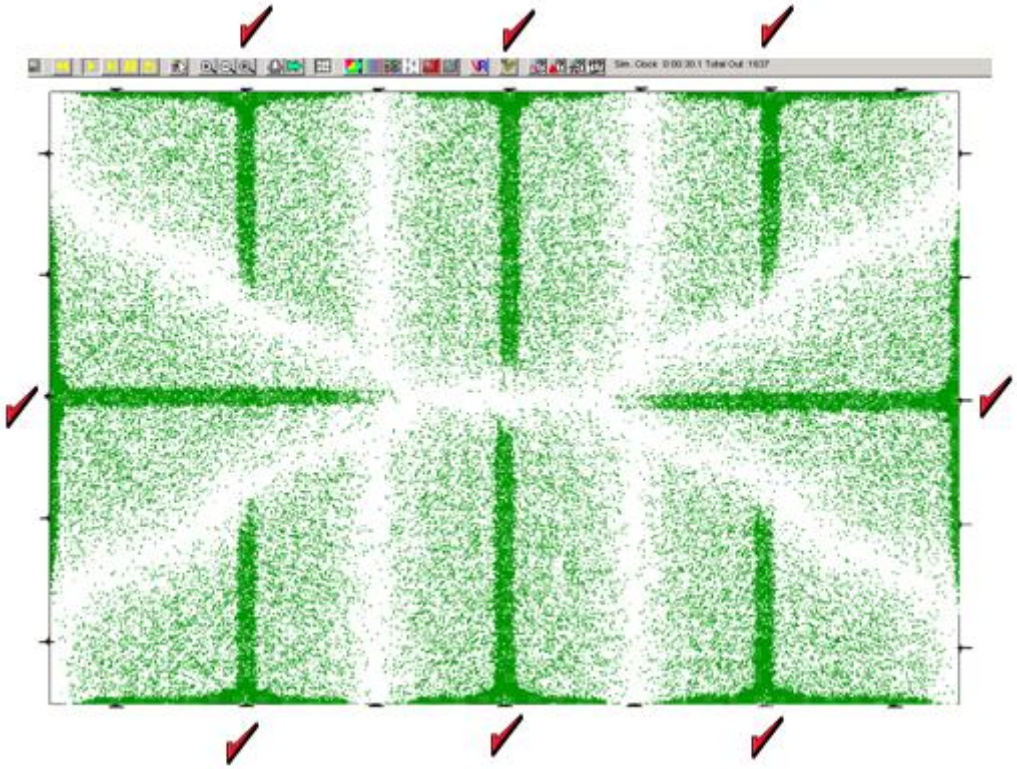


**Figure 6.12 - Population distribution at 30s**

After 70s, the population distribution is clearly displayed as being clustered round the serviceable exits and there is a distinct separation into eight regions which can be the ideal partitions to the problem, as the communication costs will be virtually non existent.

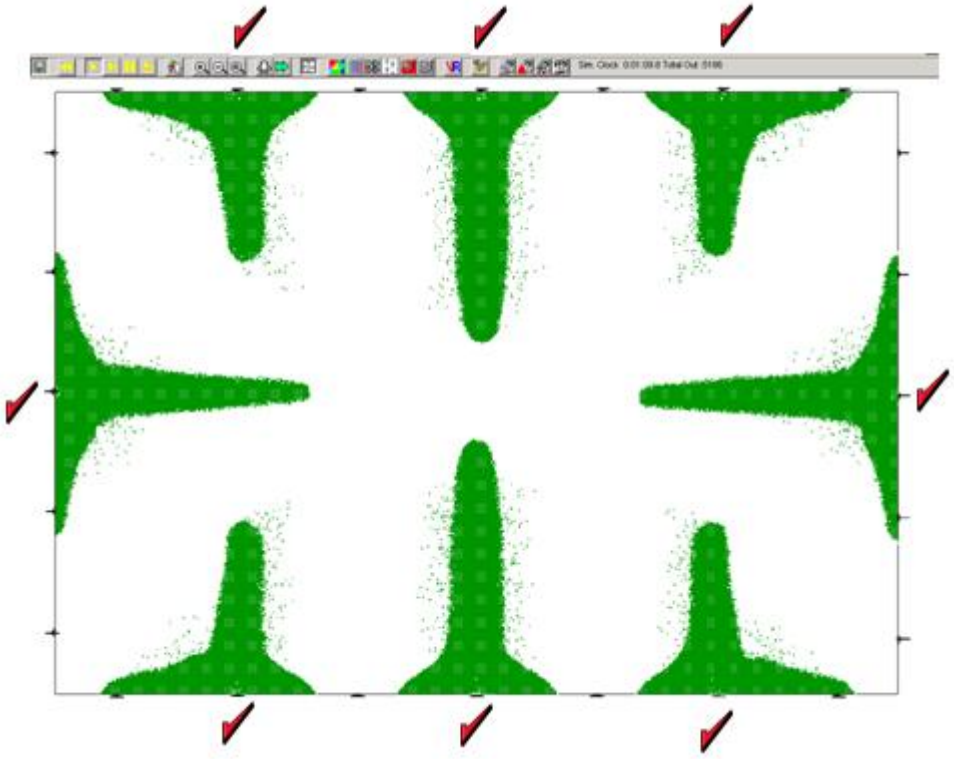


**Figure 6.13 - Population distribution at 70s**

#### 6.1.5.5.3.1 Static partitioning technique using the CPRM method on Test Case 11

The usual static partitioning algorithm employing the CPRM method was used. The CPRM method makes each exit create a sub-partition which can then be partitioned into a number of sub-parts. Hence, this strategy creates sub-domains linked to each exit and then allocates them to processors. From Figure 6.13, each of the eight processors' usage will be optimised if each one is handling one of the eight functional exits where the population distribution was concentrated. However, this method allocates the processors in exit ascending order and does not consider whether an exit is operational or not. Hence in this test case's scenario, though each processor has roughly the same number of sub-domains, they don't necessarily have an operational exit each. So this scenario involves some processors having more than one operational exit whilst others do not have any.

Figure 6.14 demonstrates the partitioned geometry when the above partitioning technique was employed on Test Case 11. Hence 24 sub-domains (for the 24 exits) were initially created when each exit does not incur sub-partitioning.

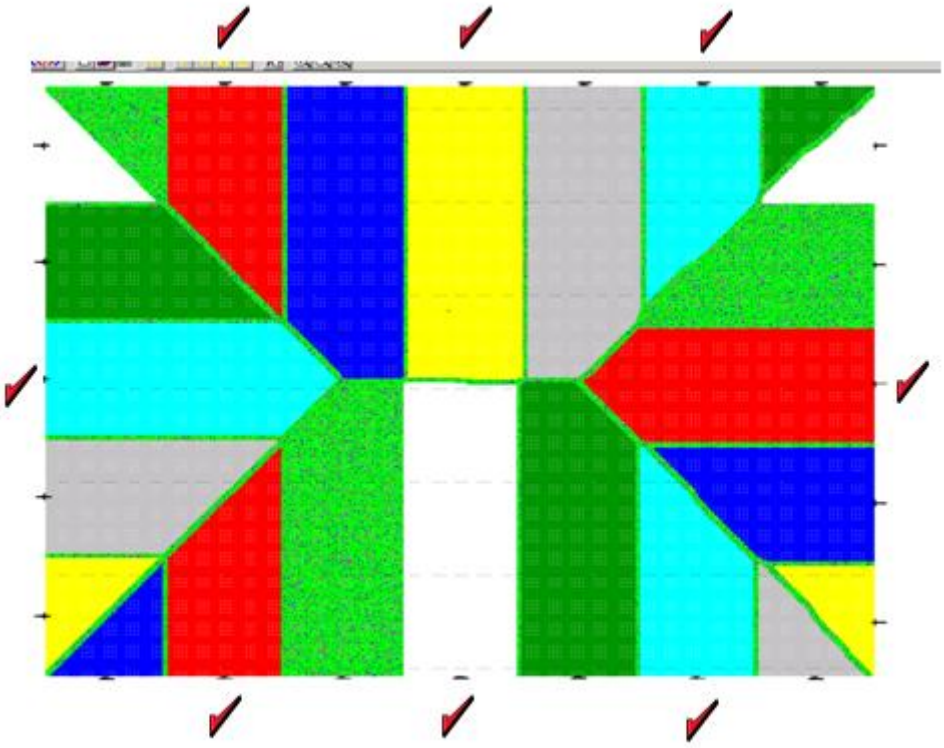


**Figure 6.14 - Initial Partition of Test Case 11**

Each colour in Figure 6.14 represents the regions serviced by a particular processor. Table 6.13 explains the colour associations to the processors and the number of serviceable exits each processor is responsible for.

**Table 6.13- Physical explanation of the partitioned Test Case 11**

| Processor Number | Colour representation | No. of exits | No. of usable exits |
|---|---|---|---|
| **P0** | Light green | 3 | 0 |
| **P1** | Red | 3 | 3 |
| **P2** | Blue | 3 | 0 |
| **P3** | Yellow | 3 | 1 |
| **P4** | Grey | 3 | 0 |
| **P5** | Cyan | 3 | 3 |
| **P6** | Dark green | 3 | 0 |
| **P7** | White | 3 | 1 |

Having this partitioned configuration, the static method was forced to keep the partitions and run the simulation in that state. From Figure 6.13, it can be seen that the ideal partition would be to only create eight partitions round the serviceable exits and not make the closed exits creating sub-domains. This static method attempted to balance out the number of exits so that each processor gets roughly the same number of exits, which would have been a wise strategy had all the exits been in use. But the scenario with some exits closed does not favour this distribution. A balancing of the serviceable exits is more beneficial to this problem, rather than balancing the total number of exits. A simple solution to this problem is to make the partitioning technique only consider useable exits when partitioning the geometry. However, this strategy will not be able to revert back to reallocating those regions if their exits become serviceable again since the static partitioning strategy does not contain this intelligent dynamic factor.

The tests were run when each exit created no extra sub-parts (24 sub-domains), two sub-parts each (48 sub-domains) and finally four sub-parts each (96 sub-domains). Figure 6.15 depicts when each exit further created four sub-parts each. Notice the highlighted partition which is sub-partitioned into four sub-parts and the corresponding partition when it was not sub-partitioned in Figure 6.14.

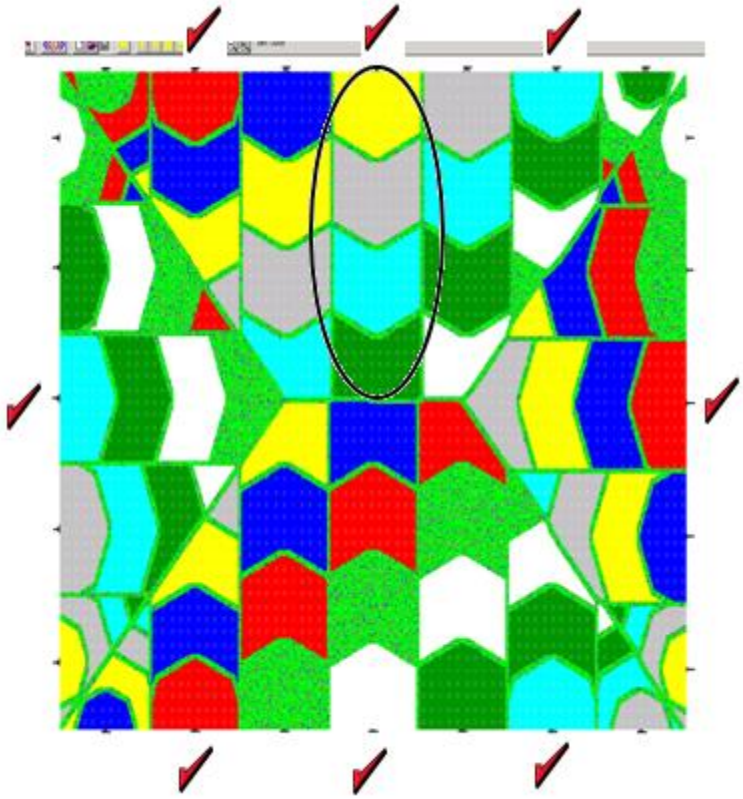


**Figure 6.15 - Test Case 11 partitioned using the CPRM method with 4 sub-parts at each exit.**

The average predicted simulation time to evacuate a population of 80,000 from this geometry is 1055.5s.The results obtained are shown in Table 6.14 .

**Table 6.14 - Speedup and efficiency values when the static partitioning strategy was used for Test Case 11**

| No. of sub-parts at each exit | Total No. of sub-domains | Speedup | Efficiency |
|---|---|---|---|
| **1** | 24 | 3.357 | 41.96% |
| **2** | 48 | 3.324 | 41.55% |
| **4** | 96 | 3.149 | 39.36% |

A maximum speedup value of only 3.357, giving a 41.96% efficiency, could be achieved when eight processors were used. This is less than half the possible speedup value of 8 achievable. However, this result is not totally unexpected as only half the processors used (i.e 4) have sub-

domains containing usable exits, which would make them busy till the end of the simulation. The other remaining four processors would be idle as soon as their local population loads move on to the sub-domains serviced by the working processors.

An interesting result is when each sub-domain was sub-partitioned into two and then four sub-parts. By having this sub-partitioning of the sub-domains, the chance of belonging to a region containing the serviceable exit is increased, giving the otherwise idle processors a chance to work for longer than was previously achieved. However, the speedup values obtained in those instances were lower than when these processors were idle. This is very surprising as the expected results were supposed to be better than when each sub-domain was not sub-partitioned further. The only possible explanation for these worse results is the increase in communication costs when more domains are created.

However, these problems can be resolved by using a dynamic reallocation algorithm. By constantly monitoring the load balancing situation of the parallel system, the dynamic reallocation strategy will try to balance out the total load amongst the processors from the parallel system.

#### 6.1.5.5.3.2 Dynamic reallocation technique using the CPRM method on Test Case 11

To verify the assumptions that having a dynamic reallocation algorithm will resolve the drawbacks that an evacuation simulation can pose (as explained in Section 6.1.5.5.3), Test Case 11 was used to demonstrate the cababilities of this algorithm . The original partitions calculated by the initial static strategy are created. As soon as the dynamic algorithm detects a load imbalance and forecasts a better reallocation of the problem, the whole geometry is reallocated and the simulation continues. Figure 6.16 shows the reallocated geometry when the dynamic reallocation algorithm detected a better partitioned problem.

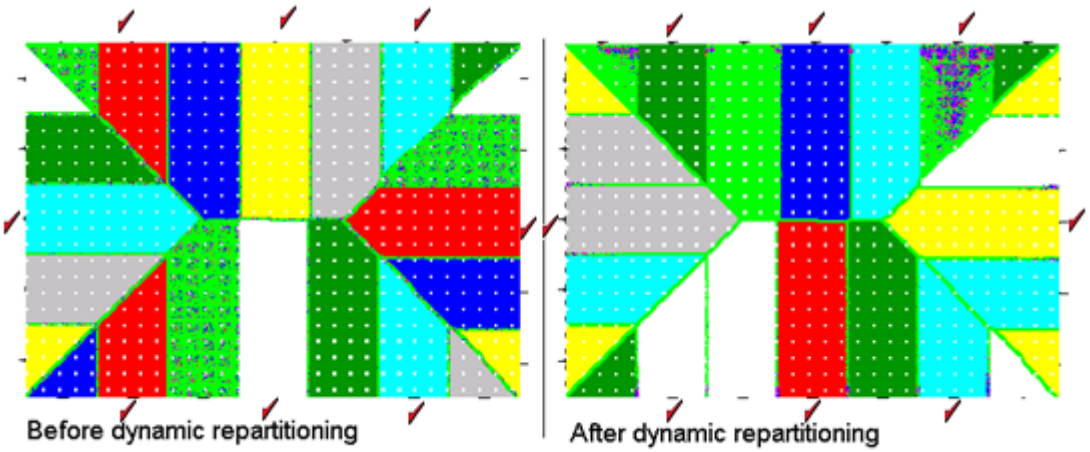



**Figure 6.16 - Test Case 11 being reallocated by the dynamic algorithm**

As can be seen by Figure 6.16, each processor (represented by their own unique colour) before repartitioning may have three usable exits (red and cyan), one exit (white and yellow) or no exits (blue, grey, light green and dark green). After repartitioning, each processor now has a single serviceable exit in its domain. Table 6.15 summarises the geometric partitioning of the new problem.

**Table 6.15 - Physical explanation of the partitioned Test Case 11 after being dynamically reallocated**

| Processor Number | Colour representation | No. of exits | No. of usable exits |
|---|---|---|---|
| **P0** | Light green | 3 | 1 |
| **P1** | Red | 1 | 1 |
| **P2** | Blue | 1 | 1 |
| **P3** | Yellow | 5 | 1 |
| **P4** | Grey | 2 | 1 |
| **P5** | Cyan | 4 | 1 |
| **P6** | Dark green | 4 | 1 |
| **P7** | White | 4 | 1 |

Now each processor has one serviceable exit and a random number of closed ones. The dynamic reallocation algorithm favours load balancing more than equally distributing the number of exits to the processors. This new reallocated configuration should make the system load balanced near to the end of the simulation. This should provide better results than when previously tested with the static technique.

The average predicted simulation time to evacuate a population of 80,000 from this geometry is 1055.5s. Table 6.16 summarises the results obtained and compares it to the results obtained previously.

**Table 6.16 - Results obtained from using the dynamic reallocation algorithm on Test Case 11**

| No. of sub-parts at each exit | Total No. of sub-domains | Speedup | | % change | Efficiency | | Average No. of reallocations |
|---|---|---|---|---|---|---|---|
| | | Static | Dynamic | | Static | Dynamic | |
| **1** | 24 | 3.357 | 6.596 | 96.48% | 41.96% | 82.45% | 10.667 |
| **2** | 48 | 3.324 | 6.457 | 94.25% | 41.55% | 80.71% | 30 |
| **4** | 96 | 3.149 | 4.566 | 45.00% | 39.36% | 57.08% | 132 |

A maximum speedup value of 6.596, with an efficiency of 82.45%, is very encouraging for a parallel system of eight processors. The dynamic repartition strategy nearly doubles the results obtained from the static method. The disappointing result of 4.566 is when each exit sub-partitioned its area into four subparts, resulting in creating 96 subparts in total. The possible explanations can be associated to the communication costs incurred with a large number of domain or to the high number of repartitions (132) which were incurred. However, when each area at each exit was not sub-partitioned, the problem was reallocated on an average of 10.667. And when each sub-domain was further sub-partitioned into two sub-parts, the dynamic reallocation algorithm found on average 30 times a necessity to reallocate the problem. This scenario produced a speedup value of 6.457, which is very favourable showing an optimised use of the parallel system.

Figure 6.17 shows the comparison in the speedup values obtained from the static and dynamic methods.

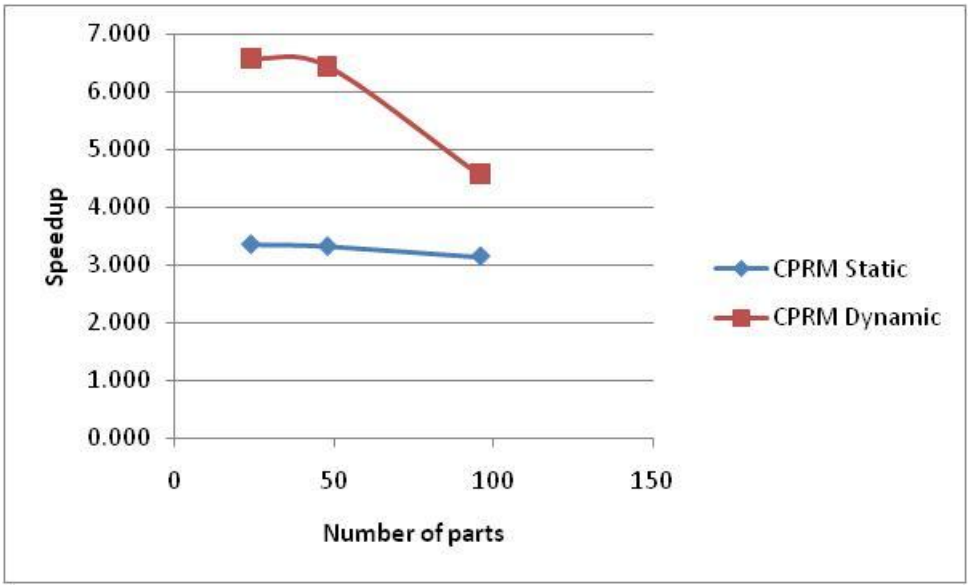



**Figure 6.17 - Comparison between static and dynamic reallocation methods using the CPRM method on Test Case 11**

As can be seen from Figure 6.17, the dynamic reallocation is a much better strategy than the static method. A vast improvement can be seen in using the dynamic reallocation algorithm from using the static version. The application of a dynamic algorithm constantly monitoring the situation of the parallel system ensures that any unpredictable evacuation scenario will be efficiently reallocated to provide a better configuration.

#### 6.1.5.5.4 METIS partitioning software on Test Case 11

The CPRM partitioning method on Test Case 11 was applied to contrive a situation which would make the static algorithm fare badly and the dynamic reallocation algorithm adapt to the contrived scenario and produced a better result. On the other hand, using the METIS partitioning software together with the static algorithm on that same Test Case should not particularly deteriorate its static run. The contrived case only worked when the CPRM method was applied.

In an attempt to make a fair comparison between the two partitioning methods, Test Case 11 was partitioned by METIS into 25, 50 and 90 parts to roughly match the number of parts created by the CPRM method (24, 48 and 96 respectively). Having an initial partition, the tests were then run in both static mode, where all the parts stay allocated to their original processors throughout

the simulation and a dynamic mode when the original partitions stay the same, but can be reallocated to a different processor.

#### 6.1.5.5.4.1 Static partitioning strategy using METIS on Test Case 11

METIS partitioned Test Case 11 into 25, 50 and 90 parts and each allocated to a processor. Eight processors were used in the parallel system to run the simulations. Each colour represents a processor and please refer to Table 6.13 for a detailed explanation of the colour coded representation. The partitioned geometries are illustrated in Figure 6.18:

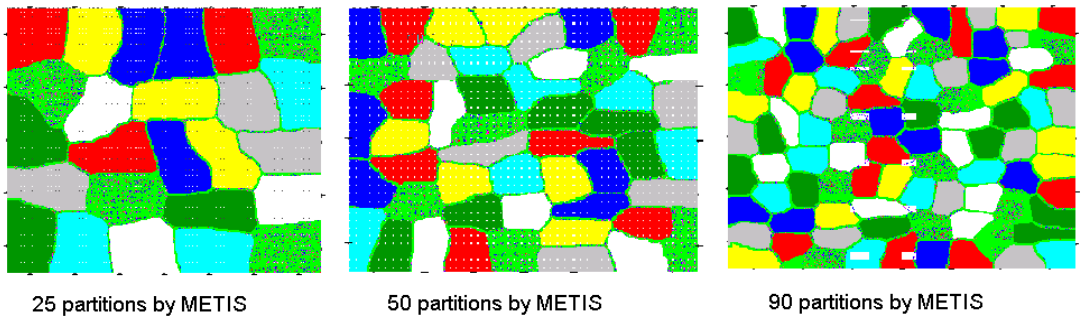



**Figure 6.18 - METIS partitioning Test Case 11 into 25, 50 and 90 parts**

The average predicted simulation time to evacuate a population of 80,000 from this geometry is 1055.5s. With the static method, the above partitions remained to their original processors throughout the simulation and the results are tabulated in Table 6.17.

**Table 6.17 - Speedup and efficiency values obtained when METIS partitioned Test Case 11 using a static method on eight processors**

| Number of sub-domains | Speedup | Efficiency |
|---|---|---|
| **25** | 4.665 | 58.31% |
| **50** | 4.302 | 53.78% |
| **90** | 3.289 | 41.11% |

A maximum speedup value of 4.665, with a 58.31% efficiency, was obtained, not surprisingly better than what was achieved when the CPRM method was used (3.357) as this was a contrived example to show a worst case scenario involving the static method. However, Test Case 11 should not be a worst case scenario when METIS was involved and thus the results obtained are not biased. So, having a maximum speedup value of 4.665 when eight processors were used is not a good indication of the performance of the parallel system.

METIS tries to minimise the number of edges created in an attempt to reduce communication costs and also tries to make the sub-domains similar in size so that the load balance is optimised. This initial partitioned geometry starts off having a balanced load and since an evacuation scenario is unpredictable in nature, the load might not stay in their original sub-domains and thus making some sub-domains redundant. So, using this static method does not guarantee a load balanced problem throughout the entire simulation.

#### 6.1.5.5.4.2 Dynamic reallocation strategy using METIS on Test Case 11

Since METIS cannot guarantee a load balanced problem it started with throughout the simulation, applying the dynamic reallocation algorithm which constantly monitors this factor should be the ideal method to use. To test this theory, the dynamic reallocation algorithm was applied to Test Case 11 and Figure 6.19 depicts the transformation from the initial partitions created by METIS to the reallocate problem.

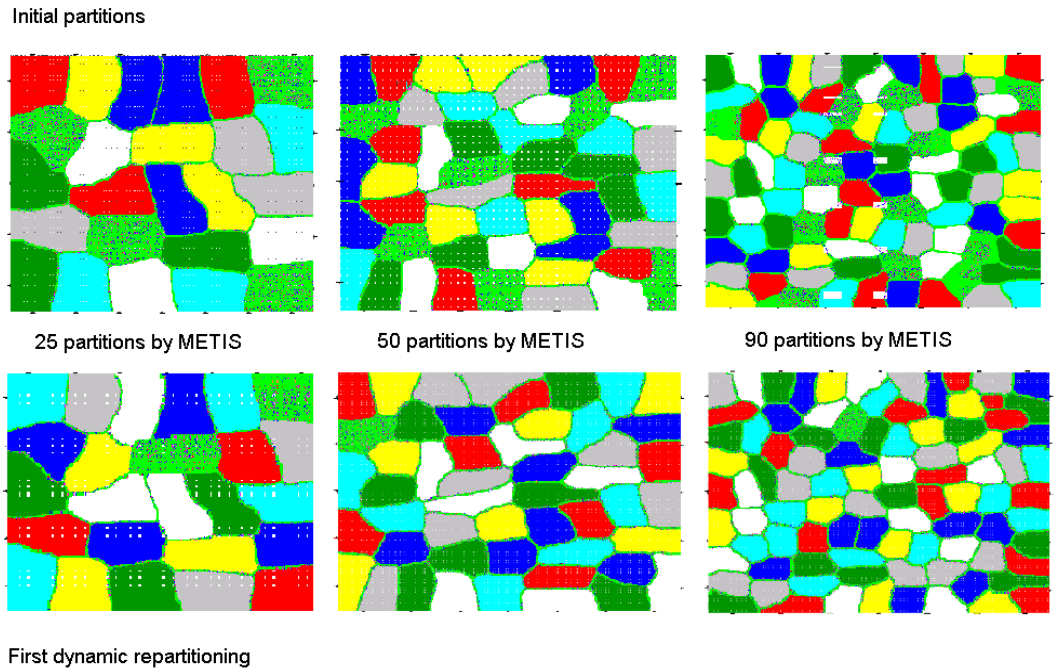



**Figure 6.19 - Test Case 11 partitioned by METIS showing the initial partitions and then the first reallocations by the dynamic reallocation algorithm**

The average predicted simulation time to evacuate a population of 80,000 from this geometry is 1055.5s. Running these cases using the dynamic reallocation algorithm gives the following results. The Speedup values obtained previously for the static case are also shown in Table 6.18 (*in italics*) for ease of comparisons.

**Table 6.18 - Speedup and efficiency values obtained when METIS partitioned Test Case 11 using the dynamic reallocation algorithm.**

| Number of sub-domains | Speedup | | % change | Efficiency | | Average Number of reallocations |
|---|---|---|---|---|---|---|
| | Static | Dynamic | | Static | Dynamic | |
| **25** | *4.665* | 5.833 | 25.04% | *58.31%* | 72.91% | 93.667 |
| **50** | *4.302* | 5.078 | 18.04% | *53.78%* | 63.48% | 158.667 |
| **90** | *3.289* | 3.748 | 13.96% | *41.11%* | 46.85% | 215.5 |

A maximum speedup value of 5.833, with 72.91% efficiency, was obtained with the use of eight processors. This case is not a contrived one like the one in 6.1.5.5.3.2 used to expose the inability

of the static method to resolve an unusual scenario in comparison to the dynamic method's ability to resolve the situation; an increase in the speedup value from 4.665 to 5.833 shows that the dynamic reallocation algorithm is also better than the static one for a normal non-contrived scenario. Even with an average number of repartitions of 93.667, which is quite high, was not a crippling factor to the speedup obtained as there was an increase from using the static version. This further emphasises how having a load imbalance (here in the static case) is worse than incurring a high reallocation cost by the dynamic reallocation technique. When the problem was partitioned into 90 parts, the dynamic reallocation algorithm found on average 215.5 occasions where there was a load imbalance with a possibility to reallocate to promote the efficiency of the parallel system. Even with this huge value of reallocations, the speedup obtained (3.748) was better than letting the system remain load imbalanced which resulted in a speedup of 3.289.

As a means of comparisons, Figure 6.20 illustrates the speedup values obtained from using both the static method and the dynamic reallocation algorithm when both the CPRM method and METIS partitions were adapted to Test Case 11.

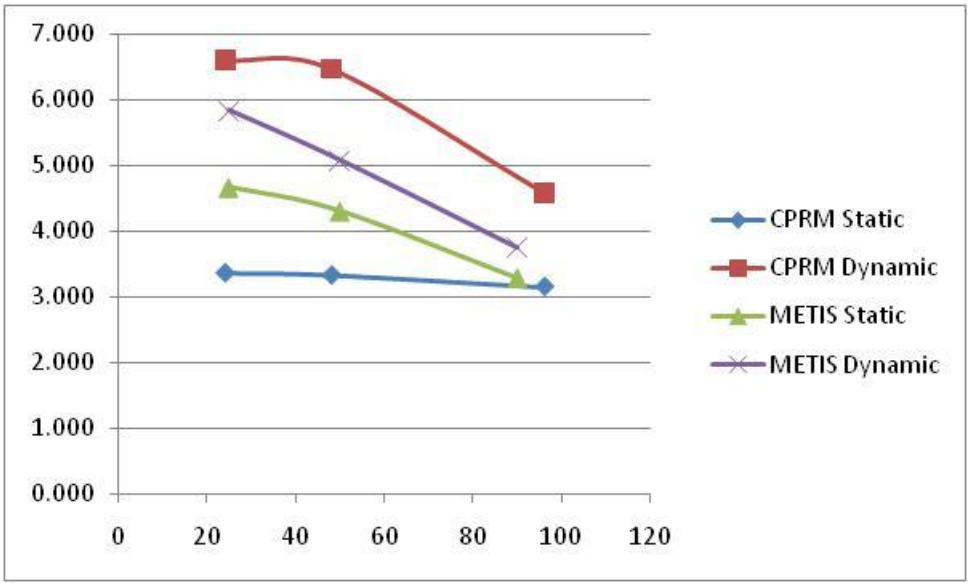



**Figure 6.20 - Comparison between static partitioning and dynamic reallocation methods using both the CPRM method and METIS on Test Case 11**

Figure 6.20 clearly demonstrates that using the dynamic reallocation algorithm is consistently better than using the static strategy. When the CPRM method was implemented, Test Case 11 was contrived to show how badly a static method would adapt to such a problem and how the dynamic reallocation strategy could adapt to the scenario and improve the reallocation of the problem. Expectedly, the resulting speedup values obtained from the dynamic strategy are much higher than its static counterpart, but also higher than those obtained when METIS was used.

#### 6.1.5.5.4.3 Idealised static partitioning algorithm on Test Case 11

An idealised partitioning can be constructed from 'Test Case 11 - Rectangular geometry measuring 500 x 700 nodes with 24 exits, 8 operational and 16 closed.

This geometry is randomly populated by 80,000 occupants. Eight processors were used in the parallel system for all cases shown in this section. Looking at Figure 6.21, it can easily be seen that partitioning this geometry along the lines that were formed as the population's paths are diverging (illustrated by the dotted lines) gives out an ideal partitioning scenario, with eight roughly similar parts being formed to be allocated to eight processors.

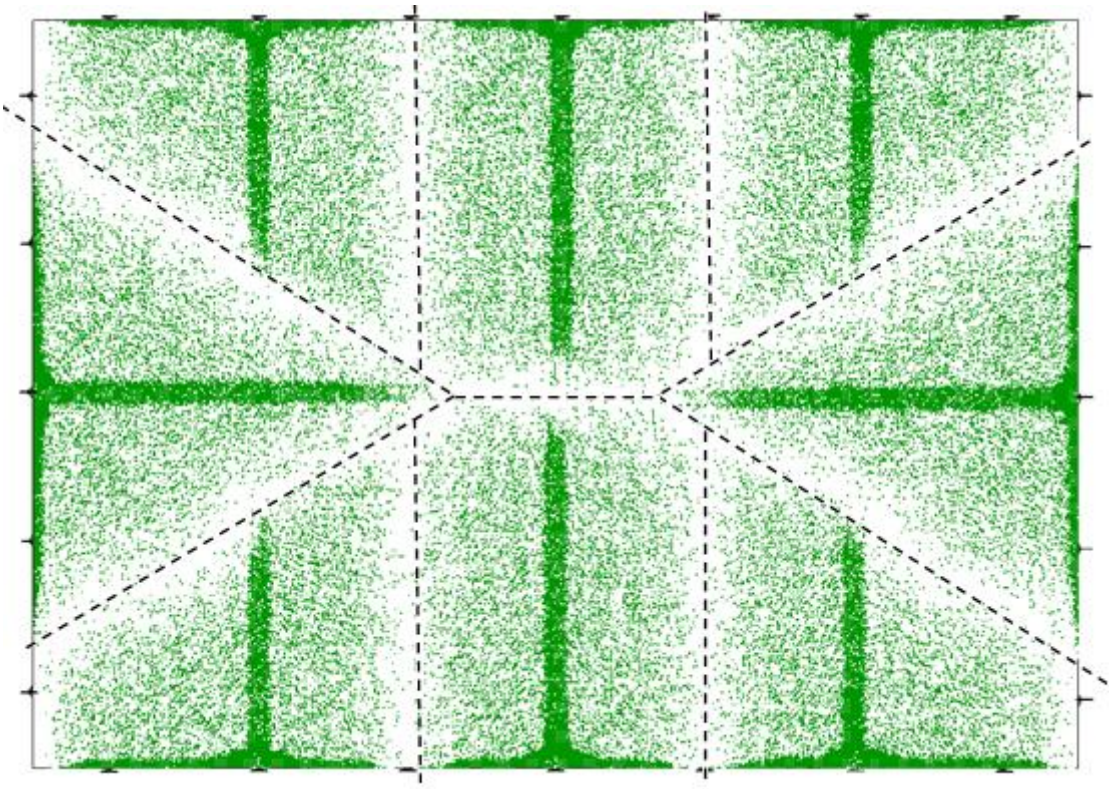


**Figure 6.21 - Ideal partition for Test Case 11**

The CPRM method can create this ideal partition if it treats the closed exits as being non-existent. It is not a good practice to ignore the closed exits as they may become active throughout the simulation. Hence, the CPRM algorithm was modified to ignore the closed exits for this Test case only. The purpose of this experiment is to test how expensive the dynamic reallocation algorithm is when applied to an idealised partitioned problem suited for a static method. Running this idealised scenario should not require any reallocation as there are eight similar sized partitions allocated to eight processors, each servicing an exit, and with this strategy the best speedup achievable is forecasted. Even though the dynamic reallocation is not expected to happen, the costs of calculating the load imbalance and the communication costs incurred at each odd simulation tick are still happening, thus adding to the total algorithmic costs. This extra cost can now be estimated by comparing the results obtained from using both the static and the dynamic algorithms.

The following tests were performed by using both the static and the dynamic algorithms; eight processors were used in the parallel system:

1. The geometry is partitioned into eight similar sub-domains containing a functional exit each. This scenario is forecasted to be the idealised case for a statically partitioned problem.

2. Using the CPRM method, each sub-domain created from the above is sub-partitioned into two sub-parts each and a total of 16 sub-domains were formed.

3. And finally, each sub-domain mentioned in 1 above is sub-partitioned into four subparts resulting in a total of 32 subparts.

The reason to create more subparts is to ascertain whether the dynamic reallocation algorithm can improve the forecasted maximum speedup expected, when the idealised case is run by using the static partitioning algorithm. Having more subparts tends to improve the load balance of a problem and merging this strategy with the dynamic algorithm which monitors this factor may give the expected improvements. Figure 6.22 illustrates the tests mentioned above.

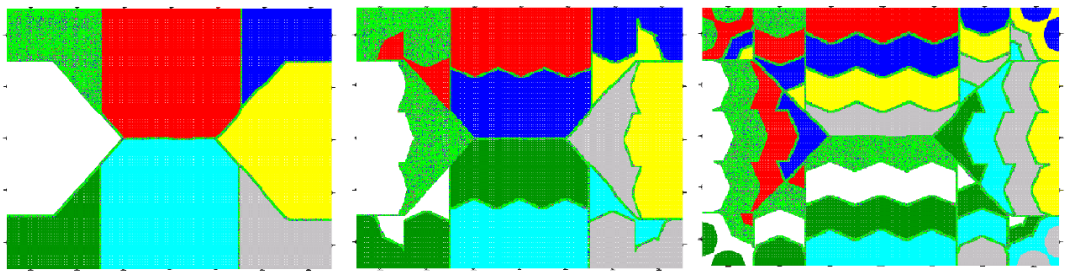



**Figure 6.22 - Partitions used on the idealised statically partitioned problem**

The average predicted simulation time to evacuate a population of 80,000 from this geometry is 1055.5s. The results obtained are shown in Table 6.19:

**Table 6.19 - Results comparing the speedup values obtained from the above tests**

| No. of sub-parts at each exit | Total No. of sub-domains | Speedup | | % change | Efficiency | | Average No. of reallocations |
|---|---|---|---|---|---|---|---|
| | | Static | Dynamic | | Static | Dynamic | |
| **1** | 8 | 6.937 | 6.847 | -1.30% | 86.71% | 85.59% | 1 |
| **2** | 16 | 6.903 | 7.137 | 3.39% | 86.29% | 89.21% | 62.667 |
| **4** | 32 | 4.574 | 4.171 | -8.81% | 57.18% | 52.14% | 26 |

As expected, the idealised statically partitioned geometry (1 sub-domain at each functional exit) produced a slightly better speedup value of 6.937 in static mode compared to the 6.847 obtained in dynamic mode. The encouraging discovery is, that the total costs incurred to monitor the load balance of the problem together with the associated communication costs at every odd simulation tick, is small as only this minor decrease in the speedup value was incurred. The number of reallocation in this case happened only once, which is the expected forced reallocation at the start of the simulation to work out the communication costs.

As stated above, the idealised static scenario was forecasted to produce the best speedup achievable for that geometrical layout. So, sub-partitioning the original sub-domains was to experiment whether this value can be further improved from the better load balancing than the increased number of sub-domains brings. By using two sub-parts at each functional exit, a speedup value of 6.903 was achieved in static mode, which is a marginal decrease from the 6.937 value obtained in the ideal case. This decrease can be associated with increased communication costs in creating more sub-domains. Though the load balance was better at the start of the simulation, with no dynamic reallocation algorithm to reallocate the processors' loads, they were stuck to having to simulate the same regions through to the end of the simulation. This expected maximum speedup value of 6.937 was improved to 7.137, when the dynamic reallocation algorithm was used on the 16 sub-domains case, confirming the assumptions that more sub-domains together with the dynamic reallocation monitoring is a good combination to adopt. When the geometry was partitioned into 16 sub-domains, an average of 62.667 reallocations still provided an improvement from 6.903 in static mode to 7.137 in dynamic mode. Again this shows

that the reallocation costs together with the algorithmic costs to determine load imbalance are negligible.

## *6.2 Concluding remarks and recommended partitioning strategy*

With the capabilities to dynamically monitor the load balancing of the problem as well as taking into considerations the network latency of the system, the dynamic reallocation strategy is clearly the method to adopt over using static partitions. The question is which partitioning method to adopt: the Cyclic Potential Route Map (CPRM) method or METIS partitioning software. Due to time constraints, more tests could not be performed to compare these two methods whilst using the dynamic reallocation strategy. However, several tests were performed to compare these two methods in static mode and the conclusion was that the CPRM does not seem to favour scalability as the number of processors were increased, unlike METIS which showed a scope for scalability. From the one test performed in dynamic mode to ascertain scalability, the same conclusions were reached as when the static strategy was applied. METIS optimises the load balance at the start of a simulation, hence creating similar sub-domains whilst minimising the number of edges in an attempt to minimise the communication costs. The CPRM method favours the load balancing of the problem, but the communication costs tends to be high. Though it was seen that the communication cost is not a determining factor compared to the load imbalance cost, a method that tries to minimise both criterion is obviously preferred. Hence, METIS is the recommended partitioning method to adopt in the partitioning of the problems.

The next question is into how many parts to partition the geometry? From Figure 6.9 in Section 6.1.5.5.2, when the problem was using a static partitioning strategy, no definite patterns could be found as to the number of sub-domains to adopt. However, when the dynamic reallocation algorithm was used, a pattern could be observed of depending on which partitioning strategy was implemented. When the CPRM method was used to partition the problem, the optimal number of sub-domains to adopt tends to increase as the number of processors was increased. METIS favoured lesser sub-domains as the number of processors was increased; the lesser processors used, the less sub-domains to adopt and as processors were being added, it was found that using lesser sub-domains provided better results. Due to time constraints and lack of more tests, no

definite value could be ascertained as to be the optimal number of sub-domains to partition a geometry. A range of 15 - 30 sub-domains provided the best results.

Whilst encouraging computational speedups were obtained to assess the performance of the parallel system, the parallel performance ratio with respect to simulation times were not fast enough to provide advice to incident commanders. The performance enhancements achieved make it viable to investigate different evacuation scenarios for planning and possibly interactive training purposes.

# 7 CONCLUSION

Large-scale events are becoming a common occurrence nowadays, making organisers ensure that adequate evacuation plans for these places are devised. Actual physical evacuations of very large areas are neither feasible nor cheap to attempt. Hence, using evacuation software to predict evacuation scenarios is a much cheaper and accommodating alternative. To gain any insight on the evacuation solutions, multiple runs and attempts need to take place so as to gain a realistic picture of the evacuation and how to go about solving the problems. An evacuation software can permit multiple runs to be done in order to gain valuable information on how to develop evacuation procedures. Another potentially important use of such a software is during live evacuation incidents. As an incident is unfolding, evacuation software could be used to evaluate the current situation to find an optimal evacuation route to successfully perform a safe and quick evacuation.

## *7.1* ***Evacuation modelling***

Evacuation modelling is well established as part of building designs to ensure performance based safety. When used on small geometries, evacuation models provide very fast results and can take hours to simulate larger areas. During live emergencies incident commanders cannot wait hours to get the evacuation results from the models. An evacuation model must be fast enough in order to permit the commanders to plan an appropriate evacuation route.

There are several evacuation models that can simulate large-scale regions and several which can only simulate smaller geometries. However, many of the models capable of simulating very large areas either provide basic evacuation times without any behavioural considerations of the evacuations. Models which can provide extra evacuation information tend to take much longer to render the results and so they are more computationally expensive. buildingEXODUS is such a model which can incorporate behavioural attributes in its evacuation simulations. Parallel

processing is a possible solution to reduce the run times of the software when large-scale evacuations are simulated.

In creating a parallel implementation of the buildingEXODUS evacuation software, some requirements were imposed by the software developers. The requirements need to satisfy the following criteria:

- There should be no change in the input or output files when using the parallel version.
- The parallel implementation is intended to function on a Microsoft Windows environment.
- Minimal additional investment is necessary by the users.
- The parallel version must work for any number of processors available, ideally connected via a local area network (LAN).
- Finally, one source code of the software including the parallel implementation is desired.

### *7.1.1 Research Objectives*

Two main research objectives were set at the start of this work and are listed below:

1. **Can parallel computing techniques be usefully applied to buildingEXODUS to enable faster wall-clock times and also to load vast cases, which cannot be loaded on a single processor?**

The development and testing of the parallel implementations of miniEXODUS and the subsequent parallelisation of buildingEXODUS demonstrated that it was feasible to gain improved computational performance.

- How much speed-up, i.e how much faster a parallel version is to its serial version, can be obtained from using the parallel techniques.

From the different partitioning methods tried on varying geometries, the majority of cases resulted in a speedup values greater than one; suggesting that the parallel version is faster than the serial one. As seen throughout this work, the speedup values obtained are highly dependent on the geometry layout used together as well as the scenario involved in the simulation. An example to relate to this finding is the fact that 10 processors provided a speedup of 15.28 when Test Case 4 (Section 5.1.1) was used and a speedup value of 3.379 when Test Case 8 (Section 5.1.5.2.5) was used to run the simulation. Hence, this objective of attaining faster wall-clock times with the parallel version is realised.

- By using all the combined memory of the different computers on the network, how big a scenario can be loaded and run?

  The initial scheme of loading different parts of a geometry on each processor was researched and tested. It is possible to load up parts of a geometry on each processor by using mini cloned nodes (Section 4.2.7.2.1) to work out which parts are on which processor. However, this method can potentially create extra data communications as the simulation is running, due to the connectivity with all the nodes within a geometry (Section 4.2.7.1).

  A solution to this setback is to use normal nodes for the partitions inherent to the processors and mini static nodes (nodes holding less attributes than a normal node as explained in 4.2.7.1) to represent the rest of the geometry. Hence, vast geometries are capable of being loaded on a processor due to the reduced attributes of the mini static nodes that make up most of the domain. In the eventuality of having such a vast problem that this strategy cannot be applied to represent the geometry on one processor, the software is able to load only parts of a geometry on each processor. However, due to time constraints more research needs to be done in this area as it has not been fully explored for any eventual problems that may arise when using this strategy.

- Is the parallel system scalable? Will increasing the number of processors on the network increase the speedup?

From the tests done on the available processors (See Chapters 5 and 6), the parallel system is definitely scalable. Due to the limited number of processors available for this research, the tests were conducted on up to 10 processors. In general whichever partitioning method is being used, as the number of processors was increased, an increasing trend was observed in the speedup achieved.

**2. Which partitioning techniques will yield the optimal partitioning strategies?**

- How to partition the problem, so that the communication costs is minimised and the load balancing is preserved?

  Whilst investigating the different partitioning methods to promote these criteria, it was found that maintaining a load balanced problem is more deterministic in getting good speedup values, rather than trying to minimise communication costs. No specific partitioning technique could be found to apply to all simulations as the nature of an evacuation model varies too much in both the geometries used as well as the different scenarios possible. Hence, a dynamic reallocation partitioning technique was eventually adopted to monitor the parallel systems for any load imbalance. This strategy inevitably incurs more communication costs but preserves the significant determining factor, which is the load balance of the problem.

- Is a generic partitioning software better suited for partitioning the problem, or must a specialised partitioning algorithm be created.

  Having devised numerous partitioning algorithms, it was found that no specific algorithm could be adopted due to the varying nature of the evacuation simulations. The two best strategies were a specially devised technique called the Cyclic Potential Route Map partitioning (CPRM) method and the partitioning software METIS. The main difference between these two techniques is that the CPRM method does not guarantee equal sized partitions compared to METIS, which promotes this feature as well as minimising communication costs. However, both methods did not guarantee good results if a static partition strategy is applied. The load balance of the problem is critically important in obtaining good speedup values and neither method can guarantee a load balanced problem throughout the simulation. After much tests in

using these two strategies, it was decided to apply METIS to devise an initial partition of the geometry and then use the specially devised dynamic reallocation algorithm as presented in Section 6.1.5.4 to monitor any load imbalance that may arise from using those partitions.

- Is dynamic repartitioning an option to consider? Will extra communication and algorithmic costs be a limiting factors on the advantages gained from a dynamic repartition?

  At the beginning of this research, it was thought that performing a dynamic repartitioning requires a huge amount of data communications and migration, which can cancel out the advantages gained by the parallel techniques. However, when running the problems with a static partitioning technique, it was found that communication cost is not a limiting factor for the parallel system. The determining factor for a good parallel performance is the load balance of the problem. Since the dynamic repartitioning technique is mostly to monitor the load balance of the problem, so it was decided to implement this strategy. A dynamic repartitioning technique could not be fully integrated due to time constraints; however, a dynamic reallocation technique was adopted. From the tests performed in Chapter 6, clearly the dynamic reallocation technique exceeded the results obtained from using the static partitioning strategy.

## 7.2 *buildingEXODUS evacuation model*

EXODUS is a suite of software designed to simulate the evacuation of people from various enclosures with buildingEXODUS part of this suite of software. buildingEXODUS simulates the evacuation of people from the built environment as well as open spaces. The enclosure is represented by a set of nodes interconnected by arcs, which permit the movement of the occupants. It uses object orientated techniques together with rule based technology to permit the movement of occupants by a set of rules and heuristics. buildingEXODUS' run-times are quick if the problem is relatively small. On the other hand, depending on the specification of the CPU used, the software may take hours to simulate larger problems and even crashes if the problem

size keeps increasing. Parallel processing can be used to reduce run times of the simulations as well as permit the evacuation from very large domains impossible to run on a single CPU.

## 7.3 Parallel implementation of the buildingEXODUS evacuation model

A parallel implementation of the buildingEXODUS model was achieved by using the master-slave parallel architecture. The domain partitioning technique was selected as it was more suited for an evacuation model as explained in Section 4.2.3.3. Each processor holds parts of a domain and consequently the occupants of those regions. The domains are static in nature and they stay on their original processor whilst the population are dynamic and can belong to several processors in the course of the simulation depending on where they are. Agents crossing over boundaries are enabled by the use of halo nodes and the message passing interface to permit the communication amongst the processors. A successful parallelisation of the buildingEXODUS software was achieved, hence honouring the first research objective.

## 7.4 Optimal partitioning strategy

The next research objective was to find an optimal partitioning strategy to divide up the problem. Since each geometry is different in layout as well as having a variety of evacuation scenarios suited to particular enclosures, an attempt to find the ideal partitioning technique which must minimise communication costs whilst preserving load balance was researched. This involved looking at both static and dynamic partitioning technique.

### 7.4.1 Static partitioning technique

A static partitioning strategy is the decomposition of a problem at the start of the problem and each part is allocated to a unique processor till the end of the simulation. Several partitioning methods were devised and tested as well as using the partitioning software METIS. Different types of geometries were used to test these partitioning methods and the two successful methods

were the Cyclic Potential Route Map (CPRM) method and METIS, depending on the geometries as well as the size of the parallel architecture. However, both methods did not indicate a scope for the scalability of the system. Since the evacuation dynamics can be unpredictable, most problems become load imbalanced as the simulation progresses, which is a crucial factor to reduce the speedup achievable. The formulation of a static partitioning strategy cannot cope with this load imbalance and thus is compelled to run with an unbalanced load. Another important aspect of an evacuation is the sudden changes to either the layouts; e.g closure of exits, or some other factors; e.g fire spread, a static partitioning formulation cannot modify its partitions to accommodate these phenomena.

### *7.4.1 Dynamic reallocation technique*

An alternative to using a static partitioning strategy was devised and a dynamic reallocation algorithm was formulated. This dynamic reallocation strategy is able to solve the load imbalance problem mentioned in the previous section as well as adapt to any sudden changes in the evacuation dynamics. This algorithm continuously monitors the load balance of the processors and reallocates the load by using the longest processing time (LPT) algorithm when load imbalance occurs. After much research was done to decide when to reallocate a problem, a gain formula was used to decide the outcome of the reallocation decisions as explained in Section 6.1.5.4. This gain formula calculates whether there is an advantage to reallocate the problem with a new balanced load compared to letting it run on its current load. Since it uses the costs of running the system at the previous time step, the gain formula is also dynamic in nature and uses only the latest costs, which can change depending on the network latency or some other synchronisation issues.

A successful dynamic reallocation strategy was devised and from the tests performed, was found to be an improvement over using its static version. From the tests done, the dynamic algorithm solved both the load imbalance issues as well as adapted to accommodate a geometry which had closed exits being serviced by some processors, which became idle when the static version was used. The dynamic reallocation algorithm reallocated the idle processors to regions having serviceable exits and thus prevented these processors from becoming idle. Hence, A novel intelligent dynamic re-allocation technique was devised so that the performance of the parallel

system is optimised. The algorithm continuously monitors the current performance of the parallel system, by using the previous time-step's parallel performance and uses a gain formula to decide whether to re-allocate the problem for a more load-balanced one.

## 7.5 Final recommendation

Having managed an improvement over using the static partitioning technique, the dynamic reallocation strategy was the final chosen method and METIS partitioning the problem into 15-30 sub-domains was found to uniformly produce scalable results. Due to time constraints, further research into a specific value for the number of sub-domains for METIS to partition the problem was not achievable. In addition, the other dynamic repartitioning strategies, like completely repartitioning the problem instead of only re-allocating the same parts to different processors, remains for further work.

Whilst encouraging computational speedups were obtained to assess the performance of the parallel system, the parallel performance ratio with respect to simulation times were not fast enough to provide advice to incident commanders. The performance enhancements achieved make it viable to investigate different evacuation scenarios for planning and possibly interactive training purposes.

## 7.6 Thesis Contribution

Several contributions were made upon the completion of the research in the domain partitioning and software modifications towards the parallelisation of the buildingEXODUS evacuation software. These contributions are listed below:

- An extensive literature review was carried out on the subjects of evacuation models, parallel computing and partitioning methods.
- This work also contributed to the parallelisation of the buildingEXODUS evacuation model via the development of miniEXODUS.

- A thorough investigation was carried out which identified the optimal partitioning strategy to adopt in the parallel version of buildingEXODUS, so that communication costs are minimised and load balance is preserved.

- A novel intelligent dynamic re-allocation technique was devised so that the performance of the parallel system is optimised. The algorithm continuously monitors the current performance of the parallel system, by using the previous time-step's parallel performance and uses a gain formula to decide whether to re-allocate the problem for a more load-balanced one.

Due to time constraints, all the avenues related to this research could not be explored. However, a comprehensive identification of which research areas to investigate was identified as future work.

# 8 FURTHER WORK

Most of the goals set at the start of this research have been achieved. This included the investigation of static partitioning strategies and the implementation of a novel dynamic load balancing system. As an outcome of this research a number of avenues could be investigated further.

## *8.1 Different parts of the geometry instead of the whole geometry on each processor*

Currently, the geometry is being partitioned to identify which sub-domains belong to which processor. However, instead of deleting the nodes which do not belong to a processor's sub-domain, all the nodes are kept but with the ones not on the sub-domain kept as static ones as explained in Section 4.2.7.2. The reason for that is because all the nodes' connectivity and interactions are linked and having the whole geometry on each processor reduces the amount of communication amongst the processors.

However, a technique was devised to only load each processor's sub-domain instead of the whole geometry as explained in Section 4.2.7.2.1. Due to a lack of time to fully explore this technique, it is not being implemented at the moment. Consequently, it is recommended to investigate this technique furtherso as to allow the simulations of very large-scale geometries to be considered.

## *8.2 Extra multi-halo regions associated with the dynamic reallocation strategy*

As explained in 6.1.1.2, extra multi-halo regions were created at the start of the simulation in order to accommodate any partitions that the dynamic reallocation algorithm renders. These multi-halo regions are not always necessary, especially if they are between two neighbouring regions serviced by the same processor, and they pose a high communication cost with respect to

the master node which is responsible to simulate those regions. This method, though not an ideal one, is still preferable than when the static partitioning strategy was used. However, a scope to improve this technique is yet to be explored.

## 8.3 *Complete repartitioning of the geometry*

At present, when dynamic reallocation occurs, the different sub-partitions stay the same but only migrate to different processors. Each sub-partition is an entity having different attributes in the program. They basically contain the nodes residing in that region and in consequence the occupants of these nodes. They are basically static in nature and nothing changes within them except to which processor they belong to. This method is easier to manage as the dynamic reallocation algorithm uses the information gained from each region to decide when to repartition. Hence, only the population within each region needs migrating and not the nodes, which greatly reduce the load of the data migration. Figure 8.1 illustrates the current dynamic reallocation strategy based on having static sub-domains.

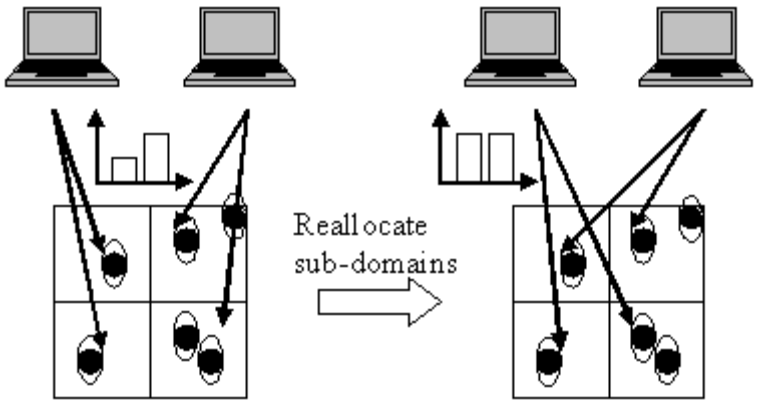



**Figure 8.1 – Illustration of static partition reallocation**

An alternative technique is the use of dynamic partitions if an imbalance occurs. The problem space is repartitioned to create a new balanced partition; see Figure 8.2.

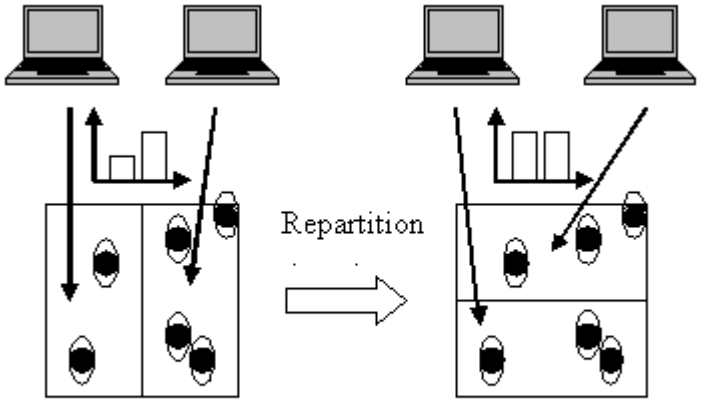



**Figure 8.2 – Illustration of dynamic repartition**

These dynamic partitions cannot be applied from the methodologies already in place for the above technique, as the sub-partitions are static in nature and do not permit the migration of the nodes as well.

Due to the time constraints, this approach could not be explored. However, it is felt that this is worthy of further examination. Upon a successful implementation of this technique, the problem of having extra multi-halo regions; see Section 6.1.1.2, might even be averted.

## *8.4 Optimal number of sub-domains to partition a geometry*

From the research performed, a definite value has not been established for the number of sub-domains required to partition a geometry and so optimise the use of a parallel system. Again, due to time constraints, an extensive investigation into that matter could not be performed. Based on the few tests performed, it was recommended that METIS partitions a domain into 15 - 30 sub-domains. However, this range of values proposed is certain to be case dependent and hence more tests are recommended to be performed to identify either a ratio or equation of the optimal number of sub-domains to partition a problem.

## 8.5 *Further Research Questions.*

As a result of this research a number of further questions have been identified.

### *8.5.1 What happens if one of the processors becomes inactive due to some hardware failure?*

- The current implementation does not cater for this situation and the whole simulation will stall if ever this scenario occurs.
- A fault tolerant parallel system must be devised in order to cater for this scenario, where the problem is repartitioned amongst the remaining working processors during run-time.
- Since the dynamic reallocation algorithm is already capable to reallocate the problem during run-time, this algorithm could be re-used to solve this problem whenever it arises.

### *8.5.2 Will the current parallel implementation of buildingEXODUS be portable for the Continuous or hybrid model of the software [36]?*

- Currently, the parallel version researched in this thesis is not portable on the continuous version of buildingEXODUS. The reasons being that:

    i. The partitioning of the geometry for the parallel runs is performed by using nodes, which the continuous version lacks. METIS uses the nodes' spatial attributes as well as their inter-connectivity in order to perform the partitioning of the problem.

    ii. The movement algorithm that was parallelised uses a nodal approach to move the population, where the nodes attract people onto it compared to

the people deciding where to move. Hence, the parallelised movement algorithm itself cannot cater for the continuous version.

- There are potentials to modify the parallel implementation to include the continuous / hybrid model as well.

    i. The method of partitioning the geometry can be performed by either using the coordinates of the geometry or using a distance / potential map that the continuous version supports. Further research needs to be done to ascertain whether METIS can still use those suggestions to partition the problem.

    ii. A key aspect of the continuous /hybrid version of buildingEXODUS is its ability to discretise a space by using a combination of macroscopic (coarse nodes) and microscopic (fine and continuous) modelling methodologies [36]. Hence, the different compartments created to represent each set of the discretised space can be used as a partition.

    iii. The current movement algorithm was successfully modified for the parallel implementation of the software, hence a similar methodology can be applied to the movement algorithm that the continuous / hybrid version in order to support any parallel implementation.

### *8.5.3 In a scenario comprising of an unrestricted number of available processors, what is the optimal number of processors to use in order to benefit from the parallel implementation?*

Unfortunately, this situation was not possible to investigate in this research, due to the limited number of processors that were available. However, as Ierotheou et al. [153] corroborated, a domain decomposition of a fixed problem size does not scale well with an increasing number of processors, hence there will come a point where adding too many processors will make the parallel system unprofitable. A possible resolution for this scenario is to allocate as many processors as the number of sub-domains created. The problem with this proposal is that a

processor managing only one sub-domain might become idle sooner than if a processor was handling several sub-domains. However, this needs to be further investigated as it is unclear what effects extra idle processors have on the parallel system.

# REFERENCES


1. Lämmel, G., Rieser, M., and Nagel, K. (2008). Bottlenecks and congestion in evacuation scenarios: A microscopic evacuation simulation for large-scale disasters. *Proc. of 7th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2008)*, Estoril, Portugal.

2. The Cabinet Office (2006). Evacuation and Shelter Guidance. Non-statutory guidance to complement Emergency Preparedness and Emergency Response & Recovery. HM Government. Available at http://interim.cabinetoffice.gov.uk/media/132739/evac_shelter_guidance.pdf . Accessed on 28th June 2011.

3. Wolshon, B., Urbina, E., Wilmot C., and Levitan, M. (2005). Review of policies and practices for hurricane evacuation. I: Transportation planning, preparedness, and response. *Natural hazards review*, **6** (2005), p.129.

4. Batty, M., Desyllas, J., and Duxbury, E. (2003). The discrete dynamics of small-scale spatial events: agent-based models of mobility in carnivals and street parades, *International Journal of Geographical Information Science*, **17**(7), pp. 673 - 697.

5. Johnson, C.W. (2006). The Application of Computational Models for the Simulation of Large-Scale Evacuations following Infrastructure Failures and Terrorist Incidents. *in Proc. NATO Research Workshop on Computational Models of Risk to Infrastructure*, pp. 9 - 13.

6. Gwynne, S., Galea, E., and Owen, M. (1997). Escape as a social response. *In Paper No. 97/IM/26, CMS Press*.

7. Shields, T. and Proulx, G. (2000), The Science of human behaviour: past research endeavours, current developments and fashioning a research agenda. *Proceedings of the sixth international symposium on fire safety science, IAFSS*, pp. 95 - 114.

8. Kuligowski, E.D. (2003). The Evaluation of a Performance-Based Design Process for a Hotel Building: The Comparison of Two Egress Models., Ph.D. Dissertation, *Department of Fire Protection Engineering, University of Maryland, College Park.*

9. Yuan, W. and Tan, K. (2007). An evacuation model using cellular automata. *Physica A: Statistical Mechanics and its Applications,* **384** (2), pp. 549 - 566.

10. Rogsch, C., Klingsch, W., Seyfried, A., and Weigel, H. (2007), How reliable are commercial software-tools for evacuation calculation?. *Interflam 2007 - Conference Proceedings*, pp. 235 - 245.

11. Kuligowski, E.D. and Gwynne, S.M.V. (2005). What a User Should Know When Selecting an Evacuation Model. *Fire Protection Engineering Magazine, Human Behaviour in Fire* Issue, Fall: pp. 30 – 40.

12. Kuligowski, E.D. and R.D. Peacock. (2005). A Review of Building Evacuation Models. *Report NIST TN 1471, MD: National Institute of Standards and Technology,* Gaithersburg.

13. Castle, C.J.E. (2007) Guidelines for Assessing Pedestrian Evacuation Software Applications. *Centre for Advanced Spatial Analysis (University College London): Working Paper*. p. 115. London, UK.

14. Santos, G. and Aguirre, B.E. (2004). A Critical Review Of Emergency Evacuation Simulation Models. *In: Proceedings of building occupant movement during fire emergencies,* Gaithersburg, Maryland.

15. Fang, Z., Yuan, J., Wang, Y., and Lo, S. (2008), Survey of pedestrian movement and development of a crowd dynamics model, *Fire Safety Journal*, **43** (6), pp. 459 - 465.

16. Grandison, A., Muthu, Y., Lawrence, P., and Galea, E.R. (2007). Simulating the evacuation of very large populations in large domains using a parallel implementation of the buildingEXODUS evacuation model. *Proceedings of the 11th International Fire Science and Engineering Conference, Interflam 2007,* 3-5th September 2007, Royal Holloway College, University of London, UK, Volume **1**, pp. 259 - 270. ISBN 978 0 9541216-8-6, 2007.

17. Schadschneider, A., Klingsch, W., Kluepfel, H., Kretz, T., Rogsch, C., and Seyfried, A.. (2008). Evacuation Dynamics: Empirical Results, Modeling and Applications. *Encyclopedia of Complexity and System Science.* arXiv:0802.1620v1.

18. Ko, S., Spearpoint, M., and Teo, A. (2007). Trial evacuation of an industrial premises and evacuation model comparison, *Fire Safety Journal,* **42** (2), pp. 91 - 105.

19. Shih, N., Lin, C., and Yang, C. (2000). A virtual-reality-based feasibility study of evacuation time compared to the traditional calculation method, *Fire Safety Journal,* **34** (4), pp. 377 - 391.

20. Lo, S., Fang, Z., Lin, P., and Zhi, G. (2004). An evacuation model: the SGEM package, *Fire Safety Journal,* **39** (3), pp. 169 - 190.

21. Johnson, C. (2005). Applying the lessons of the attack on the world trade center, 11th September 2001, to the design and use of interactive evacuation simulations, *Proceedings of the SIGCHI conference on Human factors in computing systems,* pp. 651 - 660.

22. Pelechano, N. and Malkawi, A. (2008). Evacuation simulation models: Challenges in modeling high rise building evacuation with cellular automata approaches, *Automation in Construction,* **17** (4), pp. 377 - 385.

23. Zarboutis, N. and Marmaras, N. (2007). Design of formative evacuation plans using agent-based simulation, *Safety Science,* **45** (9), pp. 920 - 940.

24. Gwynne, S., Galea, E.R., Owen, M., Lawrence, P.J., and Filippidis, L. (1999). A review of the methodologies used in evacuation modelling. *Fire and Materials* (John Wiley and Sons), **23** (6), pp. 383 - 388.

25. Zheng, X., Zhong, T., and Liu, M. (2009). Modeling crowd evacuation of a building based on seven methodological approaches. *Building and Environment,* **44** (3), pp. 437- 445.

26. Glinsky, E. and Wainer, G. (2002), Definition of Real-Time simulation in the CD++ toolkit. *In Proceedings of SCS Summer Computer Simulation Conference,* San Diego, CA.

27. Galea, E., Lawrence, P., Blake, S., Gwynne, S., and Westeng, H. (2004). A preliminary investigation of the evacuation of the WTC North Tower using computer simulation. *Proceedings of the 3rd International Symposium on Human Behavior in Fire*. Interscience Communications Ltd. (2004), pp. 167 – 180.

28. Galea, E. R., Lawrence, P., Blake, S. J., Dixon, A. J., and Westeng, H. (2007). The 2001 World Trade Centre evacuation. *In: Pedestrian and Evacuation Dynamics 2005. Springer Berlin.* Heidelberg, Berlin, Germany, pp. 225 - 238. ISBN 9783540470625

29. Galea, E.R., Sharp, G., Lawrence, P.J., and Holden, R. (2008). Approximating the Evacuation of the WTC North Tower Using Computer Simulation. *Journal of Fire Protection Engineering*, **18 (**2), pp. 85 - 115.

30. Galea, E.R., Gwynne, S., Lawrence, P.J., Filippidis, L., and Blackshields, D. (2000). buildingEXODUS V3.0 User Guide and Technical Manual, *Fire Safety Engineering Group,* University of Greenwich, UK.

31. Klupfel, H. and Meyer-Konig, T. (2003). Characteristics of the PedGo Software for Crowd Movement and Egress Simulation. 2nd International Conference in Pedestrian and Evacuation Dynamics (PED), University of Greenwich, London, U.K. pp. 331 - 340.

32. Hoskin K. (2004). Fire protection and evacuation procedures of stadia venues in New Zealand. *Department of Civil Engineering*, University of Canterbury, Private Bag 4800, Christchurch, New Zealand, pp. 3–7.

33. Wall, J.M. and Waterson, N.P. (2002). Predicting Evacuation Times – A Comparison of the STEPS Simulation Approach with NFPA 130. *Fire Command Studies*, **1** (1).

34. Ronald, N., Sterling, L., and Kirley, M. (2007). An agent-based approach to modelling pedestrian behaviour. *International Journal of Simulation. Systems, Science and Technology*, special issue on IT-Modelling in Logistics and Transport, **8** (1), pp. 25 - 38.

35. Legion Pedestrian Simulation. [On-line]. http://legion.com/ . Accessed on 18th July 2011.

36. Chooramun, N., Lawrence, P.J., and Galea, E.R. (2010). Implementing a Hybrid Space Discretisation Within An Agent Based Evacuation Model. *Pedestrian and Evacuation Dynamics* 2010, NIST, Maryland USA, March 8-10 2010, pp. 449 – 458, ISBN-10: 1441997245, ISBN-13: 9781441997241.

37. Kumar, V. (2002). Introduction to parallel computing. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA

38. Almasi, G.S. and A. Gottlieb. (1988). Highly parallel computing. Menlo Park, CA (USA); Benjamin-Cummings Pub. Co.

39. Grandison, AJ. (2003). Improving the regulatory acceptance and numerical performance of CFD based fire-modelling software. Ph.D. Thesis, University of Greenwich, London, UK.

40. Gourdain, N., Gicquel, L., Montagnac, M., Vermorel, O., Gazaix, M., Staffelbach, G., Garcia, M., Boussuge, J., and Poinsot, T. (2009). High performance parallel computing of flows in complex geometries: I. Methods, *Computational Science* and *Discovery* 2, 015003.

41. Kaludercic, B. (2004). Parallelisation of the Lagrangian model in a mixed Eulerian–Lagrangian CFD algorithm. *Journal of Parallel and Distributed Computing*, **64** (2), pp. 277 - 284.

42. Galea, E. and Ierotheou, C. (1992). Fire-field modelling on parallel computers. *Fire Safety Journal*, **19** (4), pp. 251 - 266.

43. He, F. and Wu, J. (2002). An efficient parallel implementation of the Everglades Landscape Fire model using checkpointing. *Parallel Computing*, **28** (1), pp.65 - 82, [doi: 10.1016/S0167-8191(01)00126-0].

44. Radi, B. and Estrade, J. (1998). Adaptive parallelization techniques in global weather models. *Parallel Computing*, **24** (9), pp.1167 - 1175, [doi: 10.1016/S0167-8191(98)00051-9].

45. Marrocu, M., Scardovelli, R., and Malguzzi, P. (1998). Parallelization and performance of a meteorological limited area model. *Parallel Computing*, **24** (5-6), pp. 911 - 922.

46. Anderson, D.P., Cobb, J., Korpela, E., Lebofsky, M., and Werthimer, D. (2002). SETI@ home: an experiment in public-resource computing. Communications of the ACM (ACM), **45** (11), pp.56 - 61, [doi: 10.1145/581571.581573].

47. Fire Safety Engineering Group. [On-line]. http://fseg.gre.ac.uk/exodus/index.html. Accessed on 18th July 2011.

48. Galea, E. and Gwynne, S. (2000). Principles and practice of evacuation modelling. A collection of lecture notes for a short course prepared by the *Fire Safety Engineering Group*, London, CMS Press .

49. Johnson, C.W. and Nilsen-Nygaard, L. (2008). Extending the use of evacuation simulators to support counter-terrorism: Using models of human behaviour to coordinate emergency responses to improvised explosive devices. *Proceedings of the 26th International Conference on Systems Safety*, Vancouver, Canada, International Systems Safety Society, Unionville, VA, USA.

50. Johnson, C. (2008). Using evacuation simulations for contingency planning to enhance the security and safety of the 2012 Olympic venues. *Safety Science*, **46** (2), pp. 302 – 322, [doi: 10.1016/j.ssci.2007.05.008].

51. Oven, V. and Cakici, N. (2009). Modelling the evacuation of a high-rise office building in Istanbul. *Fire Safety Journal*, **44** (1), pp. 1 - 15.

52. Batty, M. (2003). Agent-Based Pedestrian Modelling. *Advanced Spatial Analysis*: The CASA Book of GIS, ESRI Press, Redlands, CA, pp. 81 – 105.

53. Kretz, T. (2007). Pedestrian Traffic Simulation and Experiments." PhD thesis, Universität Duisburg-Essen. urn:nbn:de:hbz: 464-20070302-120944- 7.

54. Ekman, M., Warg, F., and Nilsson, J. (2005). An in-depth look at computer performance growth. *ACM SIGARCH Computer Architecture News*, **33** (1), pp. 144 – 147, [doi: 10.1145/1055626.1055646].

55. Kareem, P.A., Noorbasha, F., and Singh, R.A. (2010). Study the Task completion Time of the Benchmarks @1GHz, 2GHz and 3GHz Processors, *e-Journal of Science and Technology*, **5** (2), pp. 15 - 22.

56. Thomas Jr, S. (2004). State Regulation of Cruise Ship Pollution: Alaska's Commercial Passenger Vessel Compliance Program as a Model for Florida. *Journal of Transnational Law & Policy*, **13** (2), pp. 533-573.

57. Sweetman, B. (2003). 600 tons of technology. *Air Transport World*, **40** (4), pp. 32-36.

58. Fruin, J. (1993). The causes and prevention of crowd disasters. *Engineering for crowd safety*, Elsevier, New York, pp. 99 – 108.

59. Algadhi, S.A.H. and Mahmassani, H. (1990). Modelling crowd behavior and movement: Application to Makkah pilgrimage. *Transportation and traffic theory, Proceedings of the 11th International Symposium on Transportation and Traffic Theory*, Yokohama, Japan, Elsevier, New York, pp. 59 – 78.

60. Schäfer, R.P., Thiessenhusen, K.U., and Wagner, P. (2002). A traffic information system by means of real-time floating-car data. *Proceedings of ITS World Congress*, pp. 11 - 14.

61. Sharma, S. and Gifford, S. (2005). Using RFID to evaluate evacuation behavior models. *Fuzzy Information Processing Society*, NAFIPS, pp. 804 - 808.

62. Gupta, A. and Yadav, P. (2004). SAFE-R: a new model to study the evacuation profile of a building, *Fire Safety Journal*, **39** (7), pp. 539 - 556.

63. MacDonald, M. (2006). STEPS—Simulation of Transient Evacuation and Pedestrian Movements—User Manual. Unpublished work.

64. Hoffmann, N.A. and Henson, D.A. (1997a). Simulating emergency evacuations in stations. Paper presented in *American Public Transit Association (APTA) Rapid Transit Conference*, Washington DC, USA.

65. Hoffmann, N.A. and Henson, D.A. (1997b). Simulating transient evacuation and pedestrian movements in stations. Paper presented at the *International Conference on Mass Transit Management*, Kuala Lumpur, Malaysia.

66. STEPS software. [On-line]. http://www.mottmac.com/skillsandservices/software/stepssoftware/. Accessed on 18th July 2011.

67. Fire safety engineering group. [On-line]. http://fseg.gre.ac.uk/leaflets/exodusleaflets.html#bEXODUS_leaflet. Accessed on 18th July 2011.

68. Gwynne, S., and Galea, E. (1997). A review of the methodologies and critical appraisal of computer models used in the simulation of evacuation from the built environment. *Society of Fire Protection Engineers*, pp. 1 – 93.

69. UK Atomic Energy Authority. (2002). A Technical Summary of the AEA Egress Code. *Technical Report* AET/NOIL/27812001/002(R), Issue 1, Warrington, UK.

70. Thompson, P., Wu, J., and Marchant, E. (1996). Modelling evacuation in multi-storey buildings with simulex. *Fire Engineers Journal*, **56**, pp. 6 - 11.

71. Thompson, P.A. and Marchant, E.W. (1995). Testing and application of the computer model 'SIMULEX'. *Fire Safety Journal*, **24** (2), pp. 149 - 166. [doi:10.1016/0379-7112(95)00020-T].

72. Thompson, P. and Marchant, E. (1995). A Computer Model for the Evacuation of Large Building Populations. *Fire Safety Journal*, **24** (2), pp. 131 - 148.

73. Pathfinder (2009). Technical Reference. Thunderhead Engineering. http://www.thunderheadeng.com/downloads/pathfinder/Pathfinder_tech_ref.pdf. Accessed 25th July 2011.

74. Xiaoping Zhen, Tingkuan Zhong, and Mengting Liu (2009). Modeling crowd evacuation of a building based on seven methodological approaches. *Building and Environment*, **44** (3), pp. 437 - 445

75. Klupfel, H. (2003). A Cellular Automaton Model for Crowd Movement and Egress Simulation. PhD thesis, University Duisburg–Essen.

76. Wei-Guo, S., Yan-Fei, Y., Bing-Hong, W., and Wei-Cheng, F. (2006). Evacuation behaviors at exit in CA model with force essentials: A comparison with social force model. *Physica A: Statistical Mechanics and its Applications*, **371** (2), pp. 658 - 666.

77. Burger, T.W. [on-line]. Intel Multi-Core Processors: Quick Reference Guide. http://www.devx.com/assets/intel/13623.pdf. Accessed on 20th July 2011.

78. Castaldo, A. and Whaley, R. (2009). Minimizing startup costs for performance-critical threading. Proceedings *of the 2009 IEEE International Symposium on Parallel & Distributed Processing*, pp.1 – 8. [Doi: 10.1109/IPDPS.2009.5161010].

79. Chai, L., Gao, Q., and Panda, D.K. (2007). Understanding the impact of multi-core architecture in cluster computing: A case study with intel dual-core system. *Proceedings of the Seventh IEEE International Symposium on Cluster Computing and the Grid*, pp. 471 - 478. [Doi:10.1109/CCGRID.2007.119].

80. Hill, M. D., and Marty, M. R. (2008). Amdahl's law in the multicore era. *IEEE Computer*, **41** (7), pp. 33 – 38.

81. Nagel, K. and Rickert, M. (2001). Parallel implementation of the TRANSIMS micro-simulation. *Parallel Computing*, **27**(12) (2001), pp. 1611 – 1639

82. Transportation Research and Analysis Computing Center . Argonne National laboratory. [On-line]. http://www.anl.gov/TRACC/Research/transims_parallelization.html. Accessed 20th July 2011.

83. Cao, J., Li, M., Huang, L., Qinsheng, R., and Li, Y. (2006). Towards Building an Intelligent Traffic Simulation Platform. *Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid* (IEEE Computer Society Washington, DC, USA), 2006.

84. Kiesling, T. and Luthi, J. (2005). Towards time-parallel road traffic simulation. *Proceedings of the 19th Workshop on Principles of Advanced and Distributed Simulation*, pp.7 - 15, June 01-03, 2005. [doi:10.1109/PADS.2005.33].

85. Cameron, G., Wylie, B., and McArthur, D. (1994). Paramics: moving vehicles on the connection machine'. *Proceedings of the 1994 ACM/IEEE Conference on Supercomputing*, ACM New York, NY, USA, pp. 291 - 300.

86. Perumalla, K.S. (2006). Parallel Execution of Region-Scale Evacuation Traffic Models. *Proceedings of the 20th Workshop on Principles of Advanced and Distributed Simulation*, p.134, May 24-26, 2006. [doi: 10.1109/PADS.2006.28].

87. Perumalla, K.S. (2006). A systems approach to scalable transportation network modelling. *Proceedings of the 38th conference on Winter simulation*, pp. 1500 - 1507, December 03-06, Monterey, California

88. Perumalla, K. and Beckerman, M. (2007). An analysis approach to large-scale vehicular network simulations. *Proceedings of the 2007 summer computer simulation conference*, pp. 1223 - 1229.

89. Quinn, M. J., Metoyer, R. A., and Hunter-Zaworski, K. (2003). Parallel Implementation of the Social Forces Model. *In Proceedings of the Second International Conference in Pedestrian and Evacuation Dynamics* (August 2003), pp. 63 - 74.

90. Karavakis, E. (2010). A distributed analysis and monitoring framework for the compact Muon solenoid experiment and a pedestrian simulation. *PhD Theses*, Brunel University School of Engineering and Design.

91. Kerridge, J., Hine, J., and Wigan, M. (2001). Agent-based modelling of pedestrian movements: the questions that need to be asked and answered. *Environment and Planning B*, **28** (3), pp. 327 - 342.

92. MassMotion. [On-line]. http://www.oasys-software.com/products/engineering/massmotion.html. Accessed on 20th July 2011.

93. Morrow, E. (2010). MassMotion: simulating human behaviour to inform design for optimal performance. *The Arup Journal* (2010), pp. 38-40.

94. Helbing, D. and P. Molnar. (1995). Social force model for pedestrian dynamics. *Math. Comput. Simul Physical Review* E (APS), **51**, pp. 4282 - 4286.

95. Dagum, L. and Menon, R. (1998). OpenMP: an industry-standard api for shared-memory programming, *IEEE Comput. Sci. Eng.*, **5** (1), pp. 46 – 55. [10.1109/99.660313].

96. Guide to parallel computing. [On-line] http://ctbp.ucsd.edu/pc/html/intro1.html. Accessed on 20th July 2011.

97. Scalable computing laboratory. Types of Parallel Computers. [On-line] http://www.scl.ameslab.gov/Projects/parallel_computing/para_types.html. Accessed on 20th July 2011.

98. Light, J., Shaw, R., and Madani, M. (2002). Parallel Computing Performance Monitoring Tool for a Cluster of Computers. In M. Driscoll and T. Reeves (Eds.), *Proceedings of World Conference on E-Learning in Corporate, Government, Healthcare, and Higher Education 2002* (pp. 2661-2662). Chesapeake, VA: AACE.

99. Banicescu, I., Carico, R., Pabico, J., and Balasubramaniam, M. (2005). Design and implementation of a novel dynamic load balancing library for cluster computing. *Parallel Computing*, **31** (7), pp. 736 - 756.

100. Goscinski, A. and Wong, A. (2008). A study of the concurrent execution of parallel and sequential applications on a non-dedicated cluster. *Parallel Computing*, **34** (2), pp. 69 - 91.

101. Silva, L. and Buyya, R. (1999). Parallel programming models and paradigms. *High Performance Cluster Computing: Architectures and Systems* 2, pp. 4 - 27.

102. Baker, M. and Buyyaz, R. (1999), Cluster Computing at a Glance, chapter 1. *High Performance Cluster Computing*, vol. 1, Buyya R (ed.). Prentice-Hall: Upper Saddle River, NJ, 1999; pp. 3–47.

103. What is GPU Computing? [on-line] http://www.nvidia.com/object/GPU_Computing.html. Accessed on 20th July 2011.

104. Perumalla, K. and Aaby, B. (2008). Data parallel execution challenges and runtime performance of agent simulations on GPUs. *Proceedings of the 2008 Spring simulation multiconference*, pp. 116 - 123.

105. Fan, Z., Qiu, F., Kaufman, A., and Yoakum-Stover, S. (2004). GPU cluster for high performance computing. *Proceedings of the 2004 ACM/IEEE conference on Supercomputing*, Pittsburgh, Pennsylvania.

106. Xiao, S. and Feng, W. (2010). Inter-block GPU communication via fast barrier synchronization. *In Proceedings of the 24th IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pp. 1 - 12, Atlanta, Georgia, USA, April 2010.

107. Randall, M. and Lewis, A. (2002). A parallel implementation of ant colony optimization. Journal of Parallel and Distributed Computing, **62**, pp. 1421 - 1432.

108. Hillis, W.D. and Steele, Jr. G.L. (1986). Data parallel algorithms. Communications of the ACM - Special issue on parallelism CACM, **29**(12), pp. 1170 – 1183.

109. Silberman, A. and Marlowe, T.J. (1996). A task graph model for design and implementation of real-time systems. *Second IEEE International Conference on Engineering of Complex Computer Systems (ICECCS'96)*, p.432.

110. Peng, H., Wang, M., and Lai, C.-H. (2007). Design of parallel algorithms for fractal video compression. *International Journal of Computer Mathematics* - Distributed Algorithms in Science and Engineering, **84** (2), pp. 193 - 202.

111. Knopp, J. and Reich M.A. (1996). Workpool Model for Parallel Computing. In Workshop on High-Level Programming Models and Supportive Environments held in conjunction wit the *IEEE International Parallel Processing Symposium*, Honolulu.

112. Downton, A.C. (1995). Speed-up trend analysis for H.261 and model-based image coding algorithms using a parallel-pipeline model. *Signal Processing: Image Communication*, **7** (4-6), pp. 489 – 502.

113. Gropp, W. and E. Lusk. (2002). Goals Guiding Design: PVM and MPI. In Proc. of the IEEE Int'l Conference on Cluster Computing (Cluster 2002), pp. 257 - 265.

114. Elts, E. (2004). Comparative analysis of PVM and MPI for the development of physical applications on parallel clusters. Saint-Petersburg State University. [On-line]. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.92.3181&rep=rep1&type=pdf.

115. *PVM* and MPI: a Comparison of Features – *CSM*. [On-line]. www.csm.ornl.gov/pvm/PVMvsMPI.ps. Accessed on 20th July 2011.

116. Chan, T.F., and Tarek P. M. (1994). Domain decomposition algorithms. *Acta Numerica*, **3** , pp 61-143. doi:10.1017/S0962492900002427

117. Plimpton, S.J., Seidel, D.B., Pasik, M.F., Coats, R.S., and Montry G.R. (2003). A load-balancing algorithm for a parallel electromagnetic particle-in-cell code. *Computer Physics Communications* (Elsevier), **152** (3), pp. 227 - 241.

118. Baiardi, F., Bonotti, A., Ferrucci, L., Ricci, L., and Mori, P. (2003). Load Balancing by Domain Decomposition: the Bounded Neighbours Approach. *Proc. of the 17th European Simulation Multi Conference*. Nottingham, UK.

119. Hatzky, R. (2006). Domain cloning for a particle-in-cell (PIC) code on a cluster of symmetric-multiprocessor (SMP) computers. *Parallel Computing* (Elsevier), **32** (4), pp. 325 - 330.

120. Hendrickson, B. and Kolda, T.G. (2000). Graph partitioning models for parallel computing. *Parallel Computing* (Elsevier), **26** (12), pp. 1519 - 1534.

121. Walshaw, C., Cross, M., and Everett, M.G. (1997). Parallel dynamic graph partitioning for adaptive unstructured meshes. *Journal of Parallel and Distributed Computing* (Citeseer), **47** (2), pp. 102 - 108.

122. Karypis, G. and Kumar, V. (1998). METIS a software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices version 4.0. *University of Minnesota, Department of Comp*.

123. Karypis, G., Schloegel, K., and Kumar, V. (1997). ParMETIS: Parallel graph partitioning and sparse matrix ordering library. *Version 1.0, Dept. of Computer Science, University of Minnesota*.

124. Walshaw, C. and Cross, M. (2007). JOSTLE: Parallel Multilevel Graph-Partitioning Software - An Overview. In *Mesh Partitioning Techniques and Domain Decomposition Techniques*, edited by F. Magoules, pp. 27-58. Civil-Comp Ltd., 2007.

125. Schloegel, K., Karypis, G., and Kumar, V. (2002). Parallel static and dynamic multiconstraint graph partitioning. *Concurrency Comput. Practice Experience*, **14** (3), pp. 219 – 240.

126. Cheng, J. and Plassmann, P. (2004). A Parallel Particle Tracking Framework for Applications in Scientific Computing. *The Journal of Supercomputing*, **28** (2), pp. 149 - 164.

127. Stijnman, M., Bisseling, R., and Barkema, G. (2001). Partitioning 3D space for parallel many-particle simulations. *Computer Physics Communications*, **149** (3), pp. 121 - 134. [doi:10.1016/S0010-4655(02)00628-8 ].

128. Walshaw, C. and Cross, M. (1999). Mesh partitioning: a multilevel balancing and refinement algorithm. *Society for Industrial and Applied Mathematics*, **22** (1), pp. 63 - 80.

129. Karypis, G. and Kumar, V. (1998). Multilevel algorithms for multi-constraint graph partitioning. *Proceedings of the 1998 ACM/IEEE conference on Supercomputing* , IEEE Computer Society Washington, DC, USA, pp. 1 - 13.

130. Peschlow, P., Honecker, T., and Martini, P. (2007). A flexible dynamic partitioning algorithm for optimistic distributed simulation. *Proceedings of the 21st International Workshop on Principles of Advanced and Distributed Simulation*, IEEE Computer Society, pp. 219 - 228.

131. Biswas, R., Das, S.K., Harvey, D., and Oliker, L. (1999). Portable parallel programming for dynamic load balancing of unstructured grid applications. *in: Proceedings of 13th International Parallel Processing Symposium and 10th Symposium on Parallel and Distributed Processing*, pp. 338 – 342.

132. Hendrickson, B. and Devine, K. (2000). Dynamic load balancing in computational mechanics. *Computer Methods in Applied Mechanics and Engineering* (Elsevier), **184**, (2-4), pp. 485 - 500.

133. Bernard, P., Gautier, T., Trystram, D., Lorraine, I., and les Nancy, V. (1999). Large scale simulation of parallel molecular dynamics. *International and 10th Symposium on Parallel and Distributed Processing*, pp. 638 - 644.

134. Thomaszewski, B. and Blochinger, W. (2007). Physically based simulation of cloth on distributed memory architectures, *Parallel Computing*, **33** (6), pp. 377 - 390.

135. Sadayappan, P., Ercal, F., and Ramanujam, J. (1998). Partitioning graphs on message-passing machines by pairwise mincut. *Information Sciences—Informatics and Computer Science: An International Journal*, **111** (1-4), pp. 223 - 237.

136. Cornwell, C., Wille, L., Wu, Y., and Sklar, F. (2001). Parallelization of an ecological landscape model by functional decomposition. *Ecological Modelling*, **144** (1), pp. 13 - 20.

137. Koradi, R., Billeter, M., and Guntert, P. (2000). Point-centered domain decomposition for parallel molecular dynamics simulation. *Computer Physics Communications*, **124** (2-3), pp. 139 - 147.

138. Minyard, T. and Y. Kallinderis. (2000). Parallel load balancing for dynamic execution environments. *Computer Methods in Applied Mechanics and Engineering* (Elsevier), **189** (4), pp. 1295 - 1309.

139. Gwynne, S., Galea, E.R., Lyster, C., and Glen, I. (2003). Analysing the Evacuation Procedures Employed on a Thames Passenger Boat Using the maritimeEXODUS Evacuation Model. *Fire Technology*, **39** (3), pp. 225-246. [DOI: 10.1023/A:1024189414319].

140. Galea, E.R., Lawrence, P., Gwynne, S., Filippidis, L., Blackshields, D., Sharp, G., Hurst, N., Wang, Z., and Ewer, J. (2003). Simulating ship evacuation under fire conditions. In: E.R. Galea, Editor, *Proc Second Int Pedestrian and Evacuation Dynamics Conference*, CMS Press, Greenwich (2003), pp. 159 – 172. [ISBN 1904521088].

141. Galea, E.R., Blake, S.J., Lawrence, P.J., and Gwynne, S. (2002). The airEXODUS Evacuation Model and its Application to Aircraft Safety. Proc Fire Safety and Survivability, Applied Vehicle Technology (AVT) Panel Symp, NATO Research and Technology Organisation, Aalborg Denmark 23-26 Sept 2002.

142. Blake, S.J., Galea, E.R, Gwynne, S., and Lawrence, P.J.(2002). Adapting the airEXODUS Prototype Cabin Crewmember Redirection Procedure for use in Real Emergency Evacuations. *Report in preparation for CMS Press publication* by The University of Greenwich, 2002

143. railEXODUS. [On-line]. http://fseg.gre.ac.uk/exodus/air.html#rail . Accessed on 26th July 2011.

144. Galea, E.R., Sharp, G., Lawrence, P.J., and Holden, R. (2008). Approximating the evacuation of the World Trade Center North Tower using computer simulation. *Journal of Fire Protection Engineering*, **18** (2), pp. 85 – 115. [DOI: http://dx.doi.org/10.1177/1042391507079343].

145. Galea, E.R., Sharp, G., Lawrence, P.J., and Dixon, A. (2007). Investigating the impact of occupant response time on computer simulations of the WTC North tower evacuation. *in: Interflam 2007, Conference Proceedings, vol. 2, 11th International Fire Science and Engineering Conference, Interscience*, London, 2007, pp. 1435 – 1442.

146. JOSTLE — graph partitioning software. [On-line]. http://staffweb.cms.gre.ac.uk/~wc06/JOSTLE/ . Accessed on 25th July 2011.

147. Karypis, G. and Kumar, V. (1999). A Fast and Highly Quality Multilevel Scheme for Partitioning Irregular Graphs. *SIAM Journal on Scientific Computing*, **20** (1), pp. 359—392.

148. METIS - Serial Graph Partitioning and Fill-reducing Matrix Ordering. [On-line]. http://glaros.dtc.umn.edu/gkhome/METIS/METIS/faq . Accessed on 25th July 2011.

149. METIS - Serial Graph Partitioning and Fill-reducing Matrix Ordering. [on-line]. http://glaros.dtc.umn.edu/gkhome/METIS/METIS/overview. Accesses on 20th July 2011.

150. Wood, D.A. and Hill, M.D. (2002). Cost-effective parallel computing. *IEEE Computer*, **28** (2), pp. 69 - 72.

151. Graham, R. (1969). Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics*, **17** (2), pp. 416 - 429.

152. Wu, B. (2005). An analysis of the LPT algorithm for the max–min and the min–ratio partition problems. *Theoretical Computer Science*, **349** (3), pp. 407 - 419.

153. Ierotheou, C.S., Lai, C.-H., Palansuriya, C.J., and Pericleous, K.A. (1998). Simulation of 2-d metal cutting by means of a distributed algorithm. *Comp. Journal*, **41**, pp. 57 – 63

154. Parallel Computing at a Glance. [On-line]. www.gridbus.org/~raj/microkernel/chap1.pdf . Accessed on 20th July 2011.

# 9 APPENDIX

## *9.1* ***Publication:***

Grandison, A., Muthu, Y., Lawrence, P. and Galea, E.R. (2007). Simulating the evacuation of very large populations in large domains using a parallel implementation of the buildingEXODUS evacuation model. *Proceedings of the 11th International Fire Science & Engineering Conference, Interflam 2007*, 3-5th September 2007, Royal Holloway College, University of London, UK, Volume 1, pp. 259 - 270. ISBN 978 0 9541216-8-6, 2007.

# Simulating the evacuation of very large populations in large domains using a parallel implementation of the buildingEXODUS evacuation model.

Angus Grandison, Yasmina Muthu (now Mohedeen), Peter Lawrence and Ed Galea
Fire Safety Engineering Group
University of Greenwich
London SE10 9LS
UK

**ABSTRACT**

Computer egress simulation has potential to be used in large scale incidents to provide live advice to incident commanders. While there are many considerations which must be taken into account when applying such models to live incidents, one of the first concerns the computational speed of simulations. No matter how important the insight provided by the simulation, numerical hindsight will not prove useful to an incident commander. Thus for this type of application to be useful, it is essential that the simulation can be run many times faster than real time. Parallel processing is a method of reducing run times for very large computational simulations by distributing the workload amongst a number of CPUs. In this paper we examine the development of a parallel version of the buildingEXODUS software. The parallel strategy implemented is based on a systematic partitioning of the problem domain onto an arbitrary number of sub-domains. Each sub-domain is computed on a separate processor and runs its own copy of the EXODUS code. The software has been designed to work on typical office based networked PCs but

will also function on a Windows based cluster. Two evacuation scenarios using the parallel implementation of EXODUS are described; a large open area and a 50 storey high-rise building scenario. Speed-ups of up to 3.7 are achieved using up to six computers, with the high-rise building evacuation simulation achieving run times of 6.4 times faster than real time.

## INTRODUCTION

The use of evacuation / pedestrian modelling is well established as part of building design to ensure that buildings meet performance based safety and comfort criteria[1,2]. Another possible use for these models is to provide live operational advice to incident commanders while disasters are actually unfolding. While there are many considerations which must be taken into account when applying such models to live incidents, one of the first concerns the computational speed of simulations. No matter how important the insight provided by the simulation, numerical hindsight will not prove useful to an incident commander. Thus for this type of application to be useful, it is essential that the simulation can be run many times faster than real time. Another potential application for evacuation models is the planning of very large scale evacuation scenarios. These may involve a large part of a town or city, such as a street festival, or may involve city sized applications such as recent example of the evacuation of New Orleans. In these applications the shear size of the geometry and the number of people involved may make the running of evacuation scenarios impractical even for planning purposes.

Evacuation models that discretise space and model individual people and their interactions[1,2] are computationally expensive. Computational overheads involved in such simulations may mean that run times for simulations involving very large populations or very large geometries (or both) may require several hours. Indeed simulations, involving extremely large geometries, such as those for urban scale applications, if at all possible, may require days.

## 9.2 Parallelisation of the EXODUS core codes by the software development team

#### 9.2.1.1 *Serial movement algorithm*

Simulation Clock is incremented by 1/12s and the ticker is incremented by one tick.

1. If the number of simulation ticks is even then the movement options for the individuals are calculated goto 1. Else if the number of simulation ticks is odd then perform the movement (continue).

2. All the individuals whose PET are less than the Simulation Clock are looped over. Their PET will be incremented when they move, i.e. the travel time to a neighbouring node or set to the Simulation Clock if they have no options to move and will wait for the next Simulation Clock tick. (some individuals may not be processed as they cannot move until other individuals have moved from a node that they may wish to move to)

3. Goto 3 until all the remaining individuals' PETs have been incremented

4. Goto 1 until all the individuals have exited the evacuation scenario.

#### 9.2.1.2 *Serial enhancements*

The most important of these enhancements was a modification to the original file loading algorithm. Internally the file loading algorithm was found to be $O(N^2)$, the loading time was proportional to the square of the number of nodes (N) in the geometry; this caused excessive loading times for large geometries, and these would typically be used with the parallel implementation. The file loading was modified and the algorithm became O(N) to $O(N^2)$. In practice the algorithm operates with O(N), meaning that the loading time was now proportional to the number of nodes in the geometry. This relatively minor modification vastly reduced the amount of time needed to load a problem file. For example one case consisting of 200,000 cells originally took over 2 hours to load and this time was reduced to less than 30 seconds with the

new loading algorithm. This change does not reduce the runtime of EXODUS but substantially reduces the initial wait time for the user to load up the geometry.

Another improvement involved optimising the list of individuals used for the movement algorithm (see Figure 9.1 for pseudo-code). In the original algorithm each individual in a list is checked for movement and if some of the occupants still needed to be moved the whole list would be revisited again and again until all the occupants had moved.

| Original Code | Revised Code |
|---|---|
| `While (Need_moving)`<br>`  Need_moving = false`<br>`  For all individuals in simulation list`<br>`  Move person`<br>`  If person still needs moving`<br>`  Need_moving = true`<br>`  Endif`<br>`  Endfor`<br>`Endwhile` | `PeopleToMove = simulation list`<br>`While (PeopleToMove list not empty)`<br>`  For all individuals in PeopleToMove list`<br>`  Move person`<br>`  If person still needs moving`<br>`  Add person to NewPeopleToMove list`<br>`  Else`<br>`  Add person to PeopleAlreadyMoved list`<br>`  Endif`<br>`  Endfor`<br>`  Clear PeopleToMove`<br>`  PeopleToMove = NewPeopleToMove`<br>`  Clear NewPeopleToMove`<br>`Endwhile` |

**Figure 9.1 – Comparison of Pseudo code for Movement Loop**

#### 9.2.1.3 *Parallel movement algorithm*

The algorithm for the parallel implementation is essentially the same as the enhanced serial one apart from the population is looped over in sub-groups related to their proximity to sub-domain boundaries. These are the high-boundary, low-boundary group, and the non-boundary groups. The simulation clock is incremented by 1/12s and the ticker is incremented by one tick. A brief description of the algorithm is described:

1. If the number of simulation ticks is even then movement options for individuals are calculated then goto 1. Else if the number of simulation ticks is odd continue.

2. The individuals on high boundaries are sent to neighbouring sub-domains whilst at the same time other individuals are received from low boundaries. Whilst the computers wait to receive / send data from one another, members of the non-boundary group are computed for potential movement until this communication is complete.

3. The individuals located around the low boundaries of a sub-domain are computed for potential movement.

4. The individuals on low boundaries are sent to neighbouring sub-domains whilst at the same time the individuals are received from high boundaries. Whilst the computers wait to receive / send data from one another, further members of the non-boundary group are computed for potential movement until this communication is complete.

5. The individuals located around the high boundaries of a sub-domain are computed for potential movement.

6. Any remaining members of the non-boundary group that were not processed during step 3 or step 5 are computed for potential movement.

7. Goto 3 until all the remaining individuals' PETs have been incremented

8. Goto 1 until all the individuals have exited the evacuation scenario.

One of the important developments in the parallel implementation of EXODUS was the ability to overlap communication with calculation. A brief fragment of pseudo-code is given in Figure 9.2 to illustrate how the code overlaps communication with computation.

```
SendMessage (data of individuals crossing/approaching boundaries)
while ( NOT (All Messages Sent)  OR no more non-boundary people   )
   Process the next 16 people from non-boundary group
endwhile
if ( NOT (All Messages Sent) ) wait until All Messages Sent
…
```

**Figure 9.2 – Pseudo-code for overlapping communication with computation**

This overlapping was critical to obtaining good speedups as without this development the parallel and thus runtime performance would have been significantly poorer. By overlapping the communication and computation, the order in which individuals are processed can be changed. Also, due to the use of pseudo-random numbers in the software for some decision making meant that no two runs were the same due to the additional randomisation now created by the network. This change in ordering is just as valid as any other particular order; but unlike the serial version of EXODUS, a run cannot be repeated to give exactly the same result. An exact repeat run could be obtained by removing the overlapping of communication and computation at the expense of run time performance; though not generally useful, it was helpful to make the software run in this mode for the purposes of development and debugging.

#### 9.2.1.4 *Synchronisation*

The communications processes mentioned in the previous sections had to be carefully co-ordinated to allow the smooth transmission of the data. It is not the case that a processor can simply send and receive messages to each other.

The communication needs to be arranged to ensure that the appropriate processor is ready to receive a message when the corresponding neighbouring processor is sending that message. Failure to correctly co-ordinate this communication can lead to a deadlock. This occurs when both processors are waiting to receive a message from each other, but they will wait indefinitely as the message will not be sent until a message is received from the other processor. This critical issue was solved by using a convention based on the sub-domain numbering, and using partial non-blocking communication. All sub-domains are uniquely labelled 0 to n-1, where n is the total number of sub-domains. This leads to the general case that each sub-domain will have

boundaries with both lower and higher numbered sub-geometries; see Figure 9.3. Although only a simple 1-dimensional partition is illustrated, this applies to any arbitrary partition. These boundaries can be classified as low (neighbouring sub-domain has a lower number) and high boundaries (neighbouring sub-domain has a higher number). In order to remain synchronised, every sub-domain initiates a 'receive' from its low boundaries and simultaneously 'sends' update messages to its high boundaries (see Figure 9.3a). All boundaries wait until they have successfully received a message from their low boundaries. Once this has been achieved, the computers initiate a 'receive' from the high boundaries while simultaneously sending update messages to the low boundaries; see Figure 9.3b.

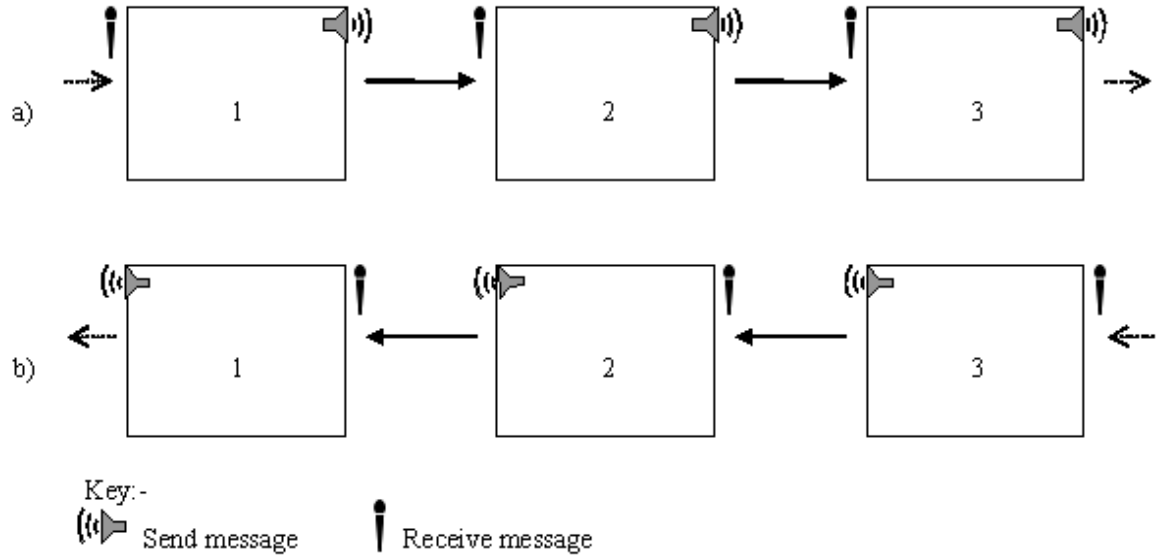



**Figure 9.3 – Communications procedure ((a) send and (b) receive) between sub-domains**

9.2.1.5 *Graphical output*

As the simulation has been distributed over a number of processors, the visualisation needs to be adapted to display the simulation on the 'Master' processor from the computations that are being calculated on the remote 'slave' processors. This is achieved by maintaining the entire nodal geometry on the 'Master' computer and updating these nodes with information from the slave processors. In order to reduce communication and memory overheads, the only information transmitted is the set of occupied nodes for each processor. This is a very fast process and does not affect the wall-clock time required to run the simulation. The graphical output is slowed

down due to the use of native Window's drawing routines and can have a substantial effect on the overall wall-clock time needed to run the simulation particularly when large population scales are involved. This problem is inherent in the serial version of EXODUS and is not specific to the parallel implementation. It is hoped that with the introduction of OpenGL/DirectX graphics (the type of graphics generally used for computer gaming and high end graphical applications) into the serial version of the software, that this issue would be mitigated in the near future. It is further expected that it will not be necessary to maintain the entire nodal geometry on the master computer in the future. The implementation of high end graphics is considered to be beyond the present scope of this work. All the testing has therefore been performed with the graphical output switched off.

## 9.3 Derivation of an estimation of the workload and the ratios for the partition cuts when N processors are used, without costs included

A rectangular geometry was considered with a free flow exit along the whole width of the geometry. The peoples' movement was assumed to resemble fluid flow. Figure 9.4 illustrates this concept.

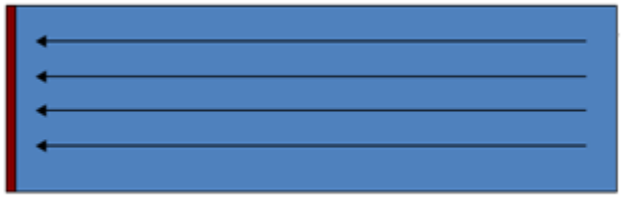


**Figure 9.4 - Geometry used to derive the partition cut formula**

*Let Z represent the population*

*l: Length of the domain*

$$Workload_{serial} = Average\ population \times Average\ distance\ travelled$$

$$Workload_{serial} = \frac{Z(1)+0}{2} \times \frac{l}{2} = \frac{Zl}{4}$$

**Serial workload**

**$T_0$-$T_3$:** $\frac{Z(1)+(0)}{2} \times \frac{l}{2} = \frac{Zl}{4}$

**2 processors**

Figure 9.5 illustrates the geometry being partitioned into two unequal parts in the ratio F1:F2

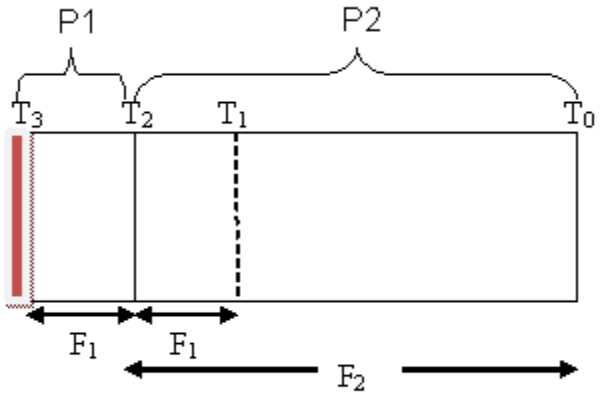



**Figure 9.5 - Geometry partitioned into two unequal parts in the ratio F1:F2**

Let Z be the initial population of the whole domain and $l$ being the length of the domain, then:

**$T_0 - T_1$:** $\frac{Z(F_2)+Z(F_1)}{2} \times \frac{(F_2-F_1)l}{2}$ (average distance)

$$= \frac{Zl(F_2+F_1)(F_2-F_1)}{4}$$

$$= \frac{Zl(F_2^2-F_1^2)}{4}$$

**$T_1$-$T_2$:** $Z(F_1) \times \frac{(F_1)l}{2} = \frac{ZlF_1^2}{2}$

**$T_2$-$T_3$:** $\frac{Z(F_1)+0}{2} \times \frac{F_1 l}{2} = \frac{Z l F_1^2}{4}$

Total workload = $\frac{Zl}{4}(F_1^2 + 2F_1^2 + F_2^2 - F_1^2)$

$= \frac{Zl}{4}(2F_1^2 + F_2^2)$

## Optimal cut: (Calculated from excel by using the in-built solver)

| F1 | F2 | Work | Max Speedup |
|---|---|---|---|
| 0.333333 | 0.666667 | 0.666667 | 1.5 |

**3 processors**

Figure 9.6 shows the geometry being partitioned into three unequal parts in the ratio F1:F2:F3

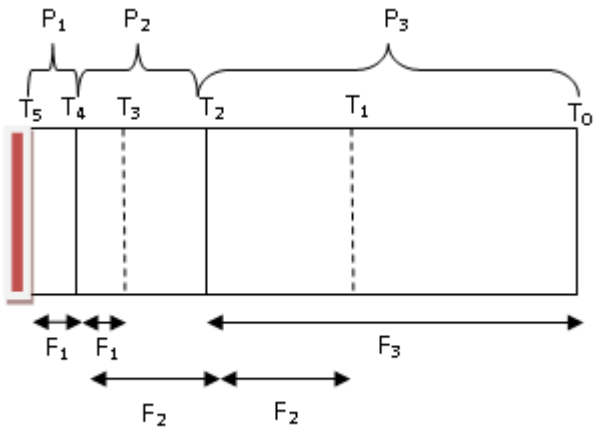



**Figure 9.6 - Geometry partitioned into three unequal parts in the ratio F1:F2:F3**

Let Z be the initial population of the whole domain and $l$ being the length of the domain, then:

**$T_0 - T_1$:** $\frac{Z(F_3)+Z(F_2)}{2} \times \frac{(F_3-F_2)l}{2}$

$= \frac{Zl(F_3+F_2)(F_3-F_2)}{4}$

$= \frac{Zl(F_3^2-F_2^2)}{4}$

**$T_1$-$T_2$:** $Z(F_2) \times \frac{(F_2)l}{2} = \frac{ZlF_2^2}{2}$

**$T_2$-$T_3$:** $\frac{Z(F_2)+Z(F_1)}{2} \times \frac{(F_2-F_1)l}{2} = \frac{Zl(F_2^2-F_1^2)}{4}$

**$T_3$-$T_4$:** $Z(F_1) \times \frac{(F_1)l}{2} = \frac{ZlF_1^2}{2}$

**$T_4$-$T_5$:** $\frac{Z(F_1)}{2} \times \frac{F_1 l}{2} = \frac{Zl}{4} F_1^2$

Total workload $= \frac{Zl}{4}(2F_1^2 + 2F_2^2 + F_3^2)$

## Optimal cut: (Calculated from excel by using the in-built solver)

| F1 | F2 | F3 | Work | Max Speedup |
|---|---|---|---|---|
| | | | | **2** |
| 0.25 | 0.25 | 0.5 | **0.5** | |

**4 processors**

Figure 9.7 illustrates the geometry being partitioned into four unequal parts in the ratio F1:F2:F3:F4.

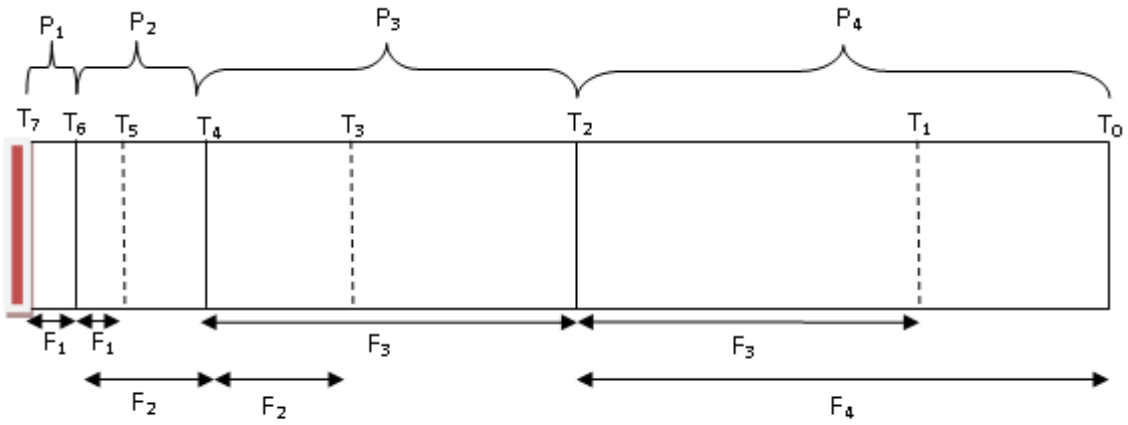



**Figure 9.7 - Geometry partitioned into four unequal parts in the ratio F1:F2:F3:F4**

Let Z be the initial population of the whole domain and $l$ being the length of the domain, then:

**$T_0 - T_1$:** $\frac{Z(F_4)+Z(F_3)}{2} \times \frac{(F_4-F_3)l}{2}$

$$= \frac{Zl(F_4+F_3)(F_4-F_3)}{4}$$

$$= \frac{Zl(F_4^2-F_3^2)}{4}$$

**$T_1$-$T_2$:** $Z(F_3) \times \frac{(F_3)l}{2} = \frac{ZlF_3^2}{2}$

**$T_2$-$T_3$:** $\frac{Z(F_3)+Z(F_2)}{2} \times \frac{(F_3-F_2)l}{2} = \frac{Zl(F_3^2-F_2^2)}{4}$

**$T_3$-$T_4$:** $Z(F_2) \times \frac{(F_2)l}{2} = \frac{ZlF_2^2}{2}$

**$T_4$-$T_5$:** $\frac{Z(F_2)+Z(F_1)}{2} \times \frac{(F_2-F_1)l}{2} = \frac{Zl(F_2^2-F_1^2)}{4}$

**$T_5$-$T_6$:** $Z(F_1) \times \frac{(F_1)l}{2} = \frac{ZlF_1^2}{2}$

**$T_6$-$T_7$:** $\frac{Z(F_1)}{2} \times \frac{F_1 l}{2} = \frac{Zl}{4} F_1^2$

Total workload = $\frac{Zl}{4}\left(2F_1^2 + 2F_2^2 + 2F_3^2 + F_4^2\right)$

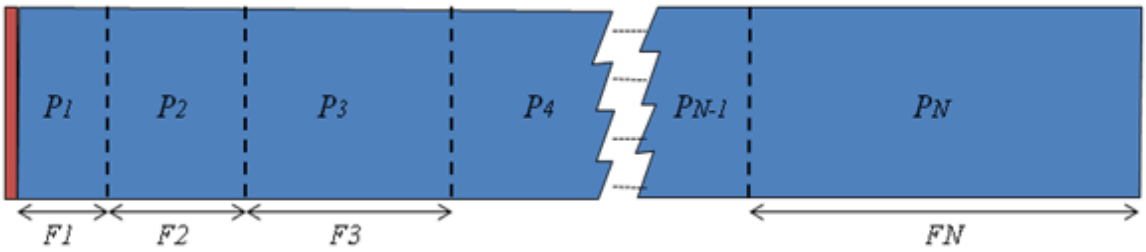
## Optimal cut: (Calculated from excel by using the in-built solver)

**4 processors**

| F1 | F2 | F3 | F4 | Max Speedup |
|---|---|---|---|---|
| 0.2 | 0.2 | 0.2 | 0.4 | **2.5** |

**N processors**

Figure 9.8 shows the geometry being partitioned into N unequal parts in the ratio F1:F2:…:FN




$$F_1 \leq F_2 \leq F_3 \leq \cdots \leq F_{N-1} \leq F_N$$

**Figure 9.8 - Geometry partitioned into N unequal parts in the ratio F1:F2:…:FN**

Let Z be the initial population of the whole domain.

By using the same principles, it can be induced that for N processors:

Total workload = $\frac{Zl}{4}\left(2\sum_{i=1}^{i=N-1} F_i^2 + F_N^2\right)$

**Optimal partitioning cut will be in the following ratios:**

Partitioning Ratio = **1:1:1:….:2**

## 9.4 Derivation of an estimation of the workload and the ratios for the partition cuts when N processors are used, with costs included

A rectangular geometry was considered with a free flow exit along the whole width of the geometry. The peoples' movement was assumed to resemble fluid flow. Figure 9.9 illustrates this concept.

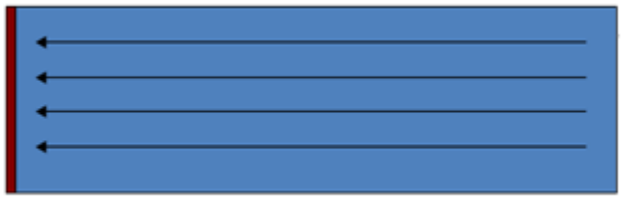


**Figure 9.9 - Geometry used to derive the partition cut formula**

*Let Z represent the population*

*l: Length of the domain*

$$Workload_{serial} = Average\ population \times Average\ distance\ travelled$$

$$Workload_{serial} = \frac{Z(1)+0}{2} \times \frac{l}{2} = \frac{Zl}{4}$$

**Serial workload**

$T_0$-$T_3$: $\frac{Z(1)+0}{2} \times \frac{(l)}{2} = \frac{Zl}{4}$

**2 processors**

Figure 9.10 shows the geometry being partitioned into two unequal parts in the ratio F1:F2.

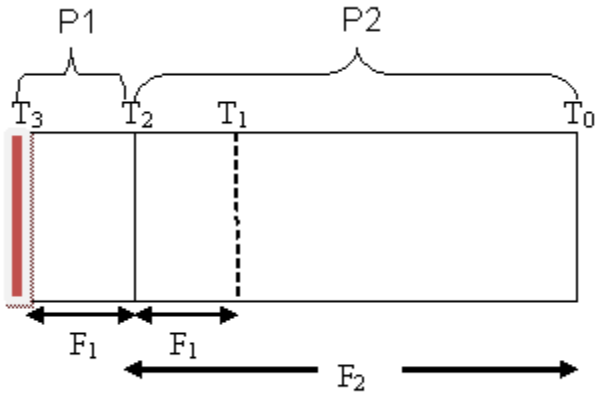



**Figure 9.10 - Geometry partitioned into two unequal parts in the ratio F1:F2**

Let Z be the initial population of the whole domain and $l$ being the length of the domain, then:

Let k be the communication costs associated with 1 person.

**$T_0 - T_1$ : P2 busier, (Comms to P1):** $\frac{Z(F_2)+Z(F_1)}{2} \times \frac{(F_2-F_1)l}{2} + Zk(F_2 - F_1)l$

$$= \frac{Zl(F_2+F_1)(F_2-F_1)}{4} + Zlk(F_2 - F_1)$$

$$= \frac{Zl(F_2^2-F_1^2)}{4} + \frac{4Zlk(F_2-F_1)}{4}$$

**$T_1$-$T_2$ : P1 busier, (comms from P2):** $Z(F_1) \times \frac{(F_1)l}{2} + Zk(F_1)l = \frac{ZlF_1^2}{2} + Zlk(F_1)$

**$T_2$-$T_3$ : P1 busier, (No comms):** $\frac{Z(F_1)+0}{2} \times \frac{F_1 l}{2} = \frac{ZlF_1^2}{4}$

Total workload = $\frac{Zl}{4}(F_1^2 + 2F_1^2 + F_2^2 - F_1^2) + \frac{Zl}{4}[4k(F_2 - F_1 + F_1)]$

$$= \frac{Zl}{4}\left(\boldsymbol{2F_1^2 + F_2^2 + 4k(F_2)}\right)$$

**Optimal cut (excluding comms): (Calculated from excel by using the built –in solver)**

| F1 | F2 | Max Speedup |
|---|---|---|
| 0.35 | 0.65 | 1.38 |

**3 processors**

Figure 9.11 illustrates the geometry being partitioned into three unequal parts in the ratio F1:F2:F3.

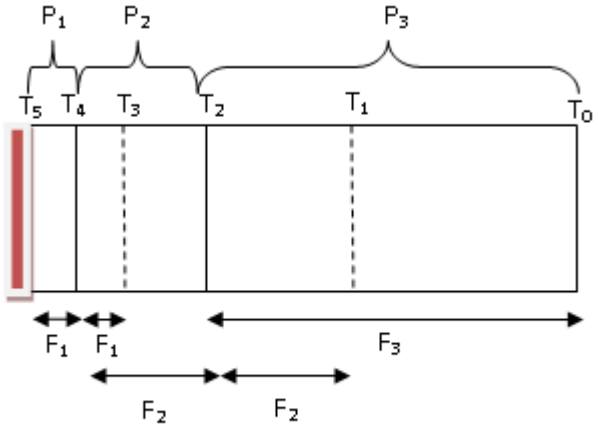



**Figure 9.11 - Geometry partitioned into three unequal parts in the ratio F1:F2:F3**

Let Z be the initial population of the whole domain and $l$ being the length of the domain, then:

**$T_0 - T_1$: P3 busier, comms to P2:** $\frac{Z(F_3)+Z(F_2)}{2} \times \frac{(F_3-F_2)l}{2} + Zl(F_3 - F_2)k$

$= \frac{Zl(F_3+F_2)(F_3-F_2)}{4} + Zl(F_3 - F_2)k$

$= \frac{Zl(F_3^2-F_2^2)}{4} + Zl(F_3 - F_2)k$

**$T_1$-$T_2$: P2 busier, comms from P3 and comms to P0:** $Z(F_2) \times \frac{(F_2)l}{2} + 2Zkl(F_2)$ =

$\frac{ZlF_2^2}{2} + 2Zlk(F_2)$

**$T_2$-$T_3$: P2 busier, comms from P2 to P1:** $\frac{Z(F_2)+Z(F_1)}{2} \times \frac{(F_2-F_1)l}{2} + Zlk(F_2 - F_1)$ =

$\frac{Zl(F_2^2-F_1^2)}{4} + Zlk(F_2 - F_1)$

**$T_3$-$T_4$: P1 busier, comms from P2:** $Z(F_1) \times \frac{(F_1)l}{2} + Zlk(F_1) = \frac{ZlF_1^2}{2} + Zlk(F_1)$

**$T_4$-$T_5$: P1 busier, no comms:** $\frac{Z(F_1)}{2} \times \frac{F_1 l}{2} = \frac{Zl}{4}F_1^2$

Total workload = $\frac{Zl}{4}\left(2F_1^2 + 2F_2^2 + F_3^2 + 4k((F_3 + 2F_2)\right)$

## Optimal cut: (Calculated from excel by using the in-built solver)

| F1 | F2 | F3 | Max Speedup |
|---|---|---|---|
| 0.26 | 0.24 | 0.5 | **1.82** |

**4 processors**

Figure 9.12 shows the geometry being partitioned into four unequal parts in the ratio F1:F2:F3:F4.

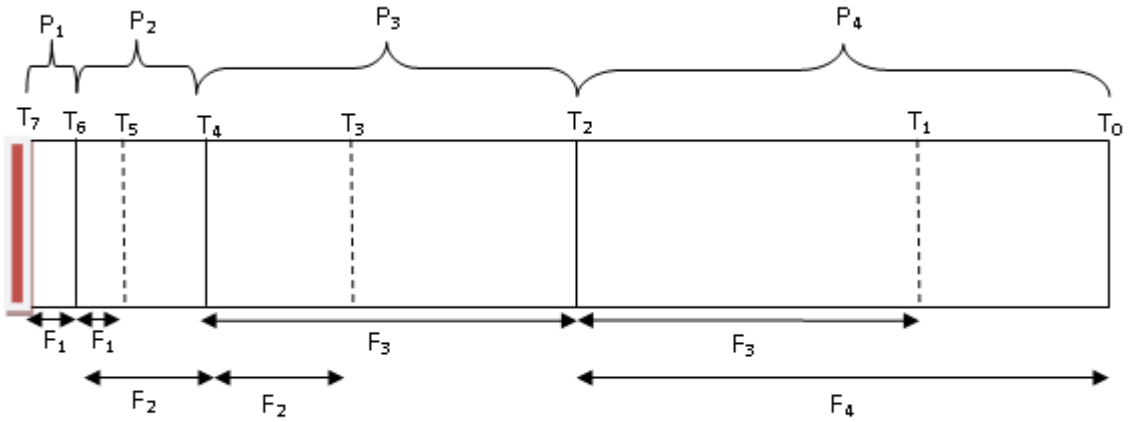



**Figure 9.12 - Geometry partitioned into four unequal parts in the ratio F1:F2:F3:F4**

Let Z be the initial population of the whole domain and $l$ being the length of the domain, then:

**$T_0 - T_1$: P4 busier, comms to P3:** $\frac{Z(F_4)+Z(F_3)}{2} \times \frac{(F_4-F_3)l}{2} + Zlk(F_4 - F_3)$

$$= \frac{Zl(F_4+F_3)(F_4-F_3)}{4} + Zlk(F_4 - F_3)$$

$$= \frac{Zl(F_4^2-F_3^2)}{4} + Zlk(F_4 - F_3)$$

**$T_1$-$T_2$: P3 busier, P3 receiving from P4 and sending to P2:** $Z(F_3) \times \frac{(F_3)l}{2} + 2Zlk(F_3) =$

$\frac{ZlF_3^2}{2} + 2Zlk(F_3)$

**$T_2$-$T_3$: P3 busier, comms to P2:** $\frac{Z(F_3)+Z(F_2)}{2} \times \frac{(F_3-F_2)l}{2} + Zlk(F_3 - F_2) =$

$\frac{Zl(F_3^2-F_2^2)}{4} + Zlk(F_3 - F_2)$

**$T_3$-$T_4$: P2 busier, P2 receiving from P3 and sending to P1:** $Z(F_2) \times \frac{(F_2)l}{2} + 2Zlk(F_2) =$

$\frac{ZlF_2^2}{2} + 2Zlk(F_2)$

**$T_4$-$T_5$: P2 busier, sending to P1:** $\frac{Z(F_2)+Z(F_1)}{2} \times \frac{(F_2-F_1)l}{2} + Zlk(F_2 - F_1) =$

$\frac{Zl(F_2^2-F_1^2)}{4} + Zlk(F_2 - F_1)$

**$T_5$-$T_6$: P1 busier, receiving comms from P2:** $Z(F_1) \times \frac{(F_1)l}{2} + Zlk(F_1) = \frac{ZlF_1^2}{2} + Zlk(F_1)$

**$T_6$-$T_7$: P1 busier, NO comms:** $\frac{Zl(F_1)}{2} \times \frac{F_1 l}{2} = \frac{Zl}{4}\boldsymbol{F_1^2}$

Total workload = $\frac{Zl}{4}\{\boldsymbol{F_4^2 + 2(F_1^2 + F_2^2 + F_3^2) + 4k}[F_4 + 2(F_3 + F_2)]\}$

**Optimal cut: (Calculated from excel by using the in-built solver)**

| F1 | F2 | F3 | F4 | Max Speedup |
|---|---|---|---|---|
| 0.21 | 0.19 | 0.19 | 0.41 | **2.20** |

**By induction, For N Processors:**

Total workload = $\frac{Zl}{4}\left\{\boldsymbol{F_N^2} + \boldsymbol{2}\sum_{i=1}^{N-1}\boldsymbol{F_i^2} + \boldsymbol{4k}[F_N + 2\sum_{i=2}^{N-1}F_i]\right\}$

## 9.5 Dr Angus Grandison's derivation of Speedup performance for a specific test case with N sub-domain and P processors

By considering the master process, which controls the black sub-domains, which will always be running at 100% and by neglecting the communication overhead; it is possible to calculate the runtime. If $l$ is the length of the domain and $z$ is the total initial population size. If there are N sub-domains and P processors then there are N/P regions consisting of sub-domains labelled 1 to P, as displayed in Figure 9.13.

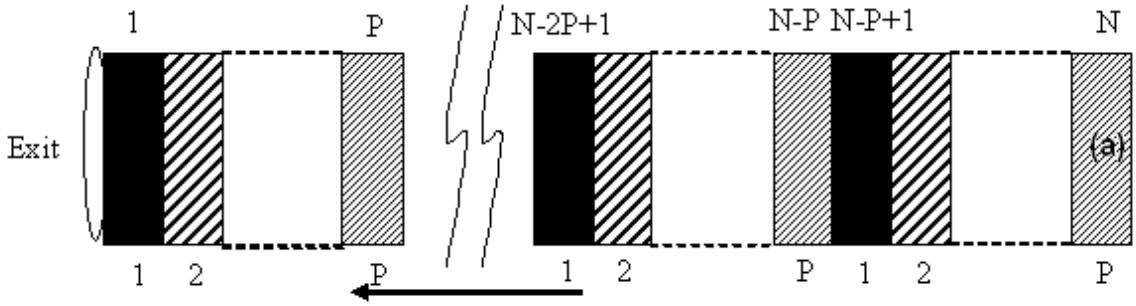



**Figure 9.13 – A generalised domain decomposition for N sub-domains and P processors**

The work done to empty (a) is distance travelled * number of people, this is represented by:

$$work_a = \frac{z}{P}\frac{l}{2N}$$

Work to empty region i = 1 is:

$$work_1 = \frac{z}{P}\frac{l}{2N}\left(P-1\right) + \frac{1}{2}\left(\frac{z}{P} + \left(\frac{z}{P} - \frac{z}{N}\right)\right)\frac{l}{2N}$$

First part of the expression is emptying from P to 2 in region i = 1 (or N to N-P+2), second part represents emptying sub-domain 1 (or N-P+1) in region i = 1.

$$work_1 = \frac{zl}{2N}\left[\frac{1}{P}\left(P-1\right)+\frac{1}{2}\left(\frac{1}{P}+\left(\frac{1}{P}-\frac{1}{N}\right)\right)\right]$$

Which becomes:-

$$work_1 = \frac{zl}{2N^2}\left[\left(\frac{N}{P}\right)\left(P-1\right)+\frac{1}{2}\left[\left(\frac{N}{P}\right)+\left(\frac{N}{P}-1\right)\right]\right]$$

For i = 2 region do the same analysis but the population size being worked on has reduced to:

$$\left(\frac{z}{P}-\frac{z}{N}\right)=\left(\frac{N}{P}-\frac{P}{P}\right)\frac{z}{N}=\left(\frac{N}{P}-1\right)\frac{z}{N}$$

To empty the i-th region the amount of work required is:

$$work_i = \frac{zl}{2N^2}\left[\left(\frac{N}{P}-\left(i-1\right)\right)\left(P-1\right)+\frac{1}{2}\left[\left(\frac{N}{P}-\left(i-1\right)\right)+\left(\frac{N}{P}-i\right)\right]\right]$$

Therefore the total work required is:

$$work_{total} = \sum_{i=1}^{N/P}\left\{\frac{zl}{2N^2}\left[\left(\frac{N}{P}-\left(i-1\right)\right)\left(P-1\right)+\frac{1}{2}\left[\left(\frac{N}{P}-\left(i-1\right)\right)+\left(\frac{N}{P}-i\right)\right]\right]\right\}$$

The work needed to empty domain with a single processor is:

$$work = \frac{zl}{4}$$

So the speedup over a single processor using P processors over N sub-domains is:

$$speedup = \frac{1}{\frac{2}{N^2}\sum_{i=1}^{N/P}\left\{\left[\left(\frac{N}{P}-\left(i-1\right)\right)\left(P-1\right)\right]+\left[\frac{1}{2}\left(\left(\frac{N}{P}-\left(i-1\right)\right)+\left(\frac{N}{P}-i\right)\right)\right]\right\}}$$

By utilising the fact that $\sum_{i=1}^{N/P} i = \frac{1}{2}\frac{N}{P}\left(\frac{N}{P}+1\right)$ and simplifying the above equation it can be shown that:

$$speedup = \frac{NP}{N+P-1}$$

## 9.6 Derivation of Speedup performance for a specific test case with N sub-domain and P processors including communication costs

By considering the master process, which controls the black sub-domains, which will always be running at 100% and by neglecting the communication overhead; it is possible to calculate the runtime. If $l$ is the length of the domain and $z$ is the total initial population size. If there are N sub-domains and P processors then there are N/P regions consisting of sub-domains labelled 1 to P, as displayed in Figure 9.14.

Let k be the communication costs associated with 1 person.

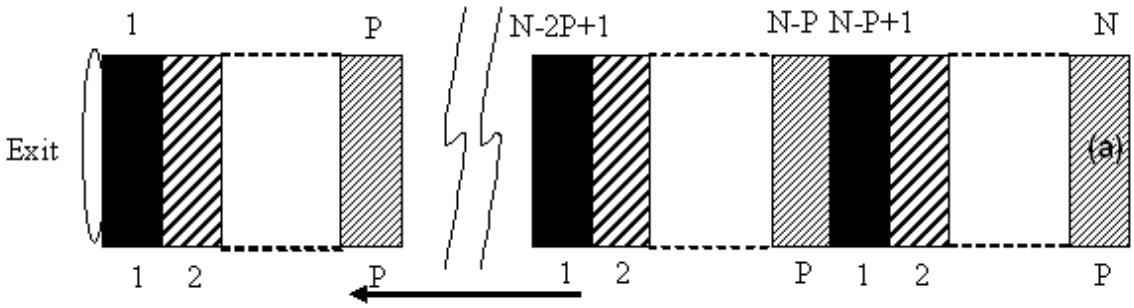



**Figure 9.14: A generalised domain decomposition for N sub-domains and P processors**

The work done to empty (a) is the average distance travelled * number of people. This is represented by:

$Work_a = \frac{\frac{l}{N}+0}{2} \times \frac{Z}{P}$ (work the master is doing across the whole geometry on same sized regions as a)

$\frac{Z}{P} = \frac{Z}{N} \times \frac{N}{P}$ (for all N/P regions)

$$Including\ communication\ costs\ of\ \ 2k\frac{Z}{P}-\frac{kZ}{N}$$

Because sending and receiving across the whole geometry from master's perspective as master is always busiest in this equal sized partitions case, the subtraction at the end is to cater for the end of the geometry where the master is not sending anymore.

$$Work_a = \frac{Z}{P}\frac{l}{2N} + 2k\frac{Z}{P} - \frac{kZ}{N}$$

(Communication costs is based on master's communication costs as the master is always busy in this equal sized partition case)

Work to empty i = 1 is:

First part of the expression is emptying from P to 2 in region i = 1 (or N to N-P+2), second part represents emptying subdomain 1 (or N-P+1) in region i = 1.

Including communication costs:

$$Work_{i=1} = \frac{Z}{P}\frac{l}{2N}(P-1) + (P-1)\frac{2kZ}{P} - (P-1)\frac{kZ}{N} + \frac{1}{2}\left(\frac{Z}{P} + \left(\frac{Z}{P} - \frac{Z}{N}\right)\right)\frac{l}{2N} + \frac{2kZ}{P} - \frac{kZ}{N} - \frac{kZ}{N}$$

$$Work_{i=1} = \frac{Zl}{2N^2}\left\{\frac{N}{P}(P-1) + \frac{1}{2}\left(\frac{2N}{P} - 1\right)\right\} + \frac{2kZ}{N}\left[\frac{N}{P}(P-1) + \frac{N}{P}\right] - \frac{kZ}{N}[(P-1)+2]$$

$$Work_{i=1} = \frac{Zl}{2N^2}\left\{\frac{N}{P}(P-1) + \frac{N}{P} - \frac{1}{2}\right\} + \frac{2kZ}{N}\left[\frac{N}{P}(P)\right] - \frac{kZ}{N}[(P+1)]$$

For i = 2 region, do the same analysis but the population size being worked on has reduced to:

$$\left(\frac{z}{P} - \frac{z}{N}\right) = \left(\frac{N}{P} - \frac{P}{P}\right)\frac{z}{N} = \left(\frac{N}{P} - 1\right)\frac{z}{N}$$

$$Work\ of\ 1\ subdomain = \left(\frac{Z}{P} - \frac{Z}{N}\right)\frac{l}{2N} + 2k\left(\frac{Z}{P} - \frac{Z}{N}\right) - \frac{kZ}{N}$$

$$= \frac{Zl}{2N^2}\left[\frac{N}{P} - 1\right] + \frac{2kZ}{N}\left(\frac{N}{P} - 1\right) - \frac{kZ}{N}$$

$$Work\ of\ (P-1) subdomains$$

$$= \frac{Zl}{2N^2}\left[\left(\frac{N}{P} - 1\right)(P-1)\right] + \frac{2kZ}{N}\left[\left(\frac{N}{P} - 1\right)(P-1)\right] - \frac{kZ}{N}(P-1)$$

$$Work\ at\ last\ (left)\ subdomain = \frac{l}{2N} \times \frac{1}{2}\left\{\left(\frac{Z}{P} - \frac{Z}{N}\right) + \left(\frac{Z}{P} - \frac{Z}{N} - \frac{Z}{N}\right)\right\} + 2k\left(\frac{Z}{P} - \frac{Z}{N}\right) - \frac{kZ}{N} - \frac{kZ}{N}$$

$$= \frac{Zl}{2N} \times \frac{1}{2}\left(\frac{2}{P} - \frac{3}{N}\right) + \frac{2kZ}{N}\left(\frac{N}{P} - 1\right) - \frac{2kZ}{N}$$

$$= \frac{Zl}{2N^2}\left[\frac{N}{P} - \frac{3}{2}\right] + \frac{2kZ}{N}\left(\frac{N}{P} - 1\right) - \frac{2kZ}{N}$$

$$Work_{i=2} = \frac{Zl}{2N^2}\left\{(P-1)\left(\frac{N}{P} - 1\right) + \frac{N}{P} - \frac{3}{2}\right\} + \frac{2kZ}{N}\left[(P)\left(\frac{N}{P} - 1\right)\right] - \frac{kZ}{N}(P+1)$$

At i = 3:

Population left per subdomain = $\frac{Z}{P} - \frac{2Z}{N}$

$$Work\ of\ 1\ subdomain = \left(\frac{Z}{P} - \frac{2Z}{N}\right)\frac{l}{2N} + 2k\left(\frac{Z}{P} - \frac{2Z}{N}\right) - \frac{kZ}{N}$$

$$= \frac{Zl}{2N^2}\left[\frac{N}{P} - 2\right] + \frac{2kZ}{N}\left(\frac{N}{P} - 2\right) - \frac{kZ}{N}$$

$$Work\ of\ (P-1) subdomains$$

$$= \frac{Zl}{2N^2}\left[\left(\frac{N}{P} - 2\right)(P-1)\right] + \frac{2kZ}{N}\left[\left(\frac{N}{P} - 2\right)(P-1)\right] - \frac{kZ}{N}(P-1)$$

$$Work\ at\ last\ (left)\ subdomain = \frac{l}{2N} \times \frac{1}{2}\left\{\left(\frac{Z}{P} - \frac{2Z}{N}\right) + \left(\frac{Z}{P} - \frac{3Z}{N}\right)\right\} + 2k\left(\frac{Z}{P} - \frac{2Z}{N}\right) - \frac{kZ}{N} - \frac{kZ}{N}$$

$$= \frac{Zl}{2N^2}\left(\frac{N}{P} - \frac{5}{2}\right) + \frac{2kZ}{N}\left(\frac{N}{P} - 2\right) - \frac{2kZ}{N}$$

$$Work_{i=3} = \frac{Zl}{2N^2}\left\{(P-1)\left(\frac{N}{P} - 2\right) + \frac{N}{P} - \frac{5}{2}\right\} + \frac{2kZ}{N}\left[\left(\frac{N}{P} - 2\right)P\right] - \frac{kZ}{N}(P+1)$$

To empty the $i^{th}$ region the amount of work required is:

$$Work_i = \frac{Zl}{2N^2}\left\{(P-1)\left(\frac{N}{P} - (i-1)\right) + \frac{N}{P} - \frac{(2i-1)}{2}\right\} + \frac{2kZ}{N}\left[\left(\frac{N}{P} - (i-1)\right)P\right] - \frac{kZ}{N}(P+1)$$

Therefore the total work required is:

$$Work_{total} = \sum_{i=1}^{\frac{N}{P}}\left\{\frac{Zl}{2N^2}\left[(P-1)\left(\frac{N}{P} - (i-1)\right) + \frac{N}{P} - \frac{(2i-1)}{2}\right]\right\} + \sum_{i=1}^{\frac{N}{P}}\frac{2kZ}{N}\left[\left(\frac{N}{P} - (i-1)\right)P\right] - \sum_{i=1}^{\frac{N}{P}}\frac{kZ}{N}[P+1]$$

$$Work_{total} = \frac{Zl}{2N^2}\sum_{i=1}^{\frac{N}{P}}\left[(P-1)\left(\frac{N}{P}-(i-1)\right)+\frac{N}{P}-\frac{(2i-1)}{2}\right]+\frac{2kZ}{N}\sum_{i=1}^{\frac{N}{P}}\left[\left(\frac{N}{P}-(i-1)\right)P\right] - \left[\frac{N}{P}\times\frac{kZ}{N}(P+1)\right]$$

$$Work_{total} = \frac{Zl}{2N^2}\sum_{i=1}^{\frac{N}{P}}\left[(P-1)\left(\frac{N}{P}-(i-1)\right)+\frac{N}{P}-\frac{(2i-1)}{2}\right]+\frac{2kZ}{N}\sum_{i=1}^{\frac{N}{P}}\left[\left(\frac{N}{P}-(i-1)\right)P\right]-\left[\frac{kZ}{P}(P+1)\right]$$

Simplifying the summation:

$$\sum_{i=1}^{\frac{N}{P}}\left[(P-1)\left(\frac{N}{P}-(i-1)\right)+\frac{N}{P}-\frac{(2i-1)}{2}\right]$$

Using the formula:

$$S_n = \frac{n(a_1+a_n)}{2}$$

$$S_{N/P} = \frac{N(a_1+a_{N/P})}{2P}$$

$$a_1 = (P-1)\left(\frac{N}{P}-0\right)+\frac{N}{P}-\left(\frac{2-1}{2}\right)$$

$$a_1 = (P-1)\left(\frac{N}{P}\right)+\frac{N}{P}-\left(\frac{1}{2}\right)$$

$$a_1 = N-\frac{N}{P}+\frac{N}{P}-\frac{1}{2}$$

$$a_1 = N-\frac{1}{2}$$

$$a_{N/P} = (P-1)\left(\frac{N}{P} - \left(\frac{N}{P} - 1\right)\right) + \frac{N}{P} - \left(\frac{\frac{2N}{P} - 1}{2}\right)$$

$$a_{N/P} = (P-1)\left(\frac{N}{P} - \frac{N}{P} + 1\right) + \frac{N}{P} - \frac{2N-P}{2P}$$

$$a_{N/P} = (P-1) + \frac{N}{P} - \frac{2N}{2P} + \frac{P}{2P}$$

$$a_{N/P} = P - 1 + \frac{N}{P} - \frac{N}{P} + \frac{1}{2}$$

$$a_{N/P} = P - \frac{1}{2}$$

$$a_1 + a_{N/P} = N - \frac{1}{2} + P - \frac{1}{2}$$

$$a_1 + a_{N/P} = N + P - 1$$

$$S_{N/P} = \frac{N(N+P-1)}{2P}$$

$$\frac{Zl}{2N^2}\sum_{i=1}^{\frac{N}{P}}\left[(P-1)\left(\frac{N}{P} - (i-1)\right) + \frac{N}{P} - \frac{(2i-1)}{2}\right] = \frac{Zl}{2N^2} \times \frac{N(N+P-1)}{2P}$$

$$= \frac{Zl}{4NP}(N+P-1)$$

$$Simplifying\ \frac{2kZ}{N}\sum_{i=1}^{\frac{N}{P}}\left[\left(\frac{N}{P} - (i-1)\right)P\right]$$

$$a_1 = \left(\frac{N}{P} - 0\right)P = N$$

$$a_{\frac{N}{P}} = \left[\frac{N}{P} - \left(\frac{N}{P} - 1\right)\right]P = P$$

$$S_{N/P} = \frac{N(N+P)}{2P}$$

$$\frac{2kZ}{N}\sum_{i=1}^{\frac{N}{P}}\left[\left(\frac{N}{P}-(i-1)\right)P\right] = \frac{2kZ}{N}\times\frac{N(N+P)}{2P} = \frac{kZ}{P}(N+P)$$

$$Work_{total} = \frac{Zl}{4NP}(N+P-1) + \frac{kZ}{P}(N+P) - \frac{kZ}{P}(P+1)$$

$$Work_{total} = \frac{Zl(N+P-1)}{4NP} + \frac{kZ}{P}(N-1)$$

The work needed to empty domain with a single processor is:

$$Work = \frac{Zl}{4}$$

So, the speedup over a single processor using P processors over N sub-domains is:

$$Speedup = \frac{\frac{Zl}{4}}{\frac{Zl(N+P-1)}{4NP} + \frac{kZ}{P}(N-1)}$$

$$Speedup = \frac{lNP}{l(N+P-1) + 4Nk(N-1)}$$

Let S represent the Speedup.

$$S = \frac{lNP}{l(N+P-1)+4Nk(N-1)} = \frac{lNP}{lN+lP-l+4kN^2-4kN}$$

Differentiating the speedup with respect to N will give the optimal N to yield the maximum speedup:

$$\frac{\partial S}{\partial N} = \frac{(lN+lP-l+4kN^2-4kN)lP-lNP(l+8kN-4k)}{[l(N+P-1)+4Nk(N-1)]^2}$$

$$\frac{\partial S}{\partial N} = \frac{l^2NP+l^2P^2-l^2P+4klPN^2-4klNP-l^2NP-8klPN^2+4klNP}{[l(N+P-1)+4Nk(N-1)]^2}$$

$$\frac{\partial S}{\partial N} = \frac{l^2(P^2-P)-4klPN^2}{[l(N+P-1)+4Nk(N-1)]^2}$$

At maximum S, $\frac{\partial S}{\partial N} = 0$

$$\frac{\partial S}{\partial N} = \frac{l^2(P^2-P)-4klPN^2}{[l(N+P-1)+4Nk(N-1)]^2} = 0$$

$$l^2(P^2-P)-4klPN^2 = 0$$

$$4klPN^2 = l^2(P^2-P)$$

$$4kN^2 = l(P-1)$$

$$N^2 = \frac{l(P-1)}{4k}$$

$$N = \sqrt{\frac{l(P-1)}{4k}}$$

## 9.7 ***Final Algorithm for dynamic reallocation***

**At Even clicks**

**{**

if SimTime = 0

{

1. LoadBalanceAlgo(Population
   - Perform LPT algo based on the population load
2. RepartitionDynamically()
   - Get the cost of updating the halos (HaloCosts)
   - Get the cost of sending 1 person (Total sendCost/NumPplSent) (SetCostSendPerPerson()), Note that this includes the deleting of the people as well
   - Get the cost of Receiving 1 person (Total sendCost/NumPplRecv) (SetCostRecvPerPerson()), Note that this include the cost of creating the people as well

}

If(SimTime > 0 and SimTime = multiple of 12)

{

1. Bool Repartition = DetermineLoadImbalance()

   DetermineLoadImbalance()

   {

   - LoadBalanceAlgo(WorkLoad)

     {

     (1) AllReduce is called to get the populationLoad and the WorkLoad in each subpartition and then the subpartitions are updated accordingly
     - Get the Maximum current workload (GlobalCurrentMaxLoad ), i.e compare the Workload of each processor and the maximum is selected
     - Broadcast the GetWorkLoadCost() of proc having the max workload to MaxWorkLoadCost

     (2) Perform LPT algo based on the WorkLoads in each processor

- Get the Maximum New workload (GlobalNewMaxLoad), i.e compare the new Workload of each processor and the maximum is selected
- NumPplSent and NumPplRecv are updated.

}

- CostCurrentLoad = MaxWorkLoadCost* GlobalCurrentMaxLoad
- CostNewLoad = MaxWorkLoadCost* GlobalNewMaxLoad
- ReshufflingCost = HaloCosts + NumPplSent *GetCostSendPerPerson() + NumPplRecv *GetCostRecvPerPerson()
- AllReduce the ReshufflingCost to MaxReshufflingCost
- Gain = CostCurrentLoad – (CostNewLoad + MaxReshufflingCost)
- If (Gain > 0)

Repartition = true

}

}

2. If (Repartition)

RepartitionDynamically()

RepartitionDynamically()
{
*Pack and send old population*
SetCostSendPerPerson()
*Disassociate local nodes and claim new domain, update halos*
HaloCosts
*Receive, unpack and create new population*
SetCostRecvPerPerson()
}

**}**

**At Odd clicks**

**{**

1. Movement(), Get the workload (TotalLoad)
2. Time Movement()
   - SetWorkLoadCost(), ( Time / TotalLoad)

**}**