International Journal of Soft Computing 11 (6): 427-436, 2016
ISSN: 1816-9503
© Medwell Journals, 2016

A Comparative Taxonomy of Parallel Algorithms for Crowd Dynamics Models and their Simulators

[1]Khalid Mohammad Jaber, [2]Mohammed Mahmod Shuaib,
[1]Randa Maraqa and [3]Osama Moh'd Alia
[1]Faculty of Science and Information Technology,
Al-Zaytoonah University of Jordan, Amman, Jordan
[2]Department of Computer Sciences, College of Sharia and Islamic Studies in Al Ahsaa, Al-Imam Muhammad Ibn Saud Islamic University (IMSIU), 31982 Al Ahsaa, Saudi Arabia
[3]Department of Computer Science, Faculty of Computers and Information Technology,
University of Tabuk, P.O. Box 741, 71491 Tabuk, Kingdom of Saudi Arabia

Abstract: Massive congestion is a very serious concern that can lead to disasters. The development of crowd dynamics models and crowd simulation tools is essential to better represent congestion aspects as well as to evaluate proposed solutions. However, model development normally involves increasing the mathematical complexity, imposing higher computational demands. This has led researchers to investigate solutions that can reduce or minimize the computational demands of such models where among them is the parallel computing approaches. In this study we highlight the application of parallel computing in reducing computational demands for crowd dynamic simulators while simultaneously improving their performances. This study also includes a comprehensive overview of the state-of-the-art parallel computing approaches used in crowd dynamic simulators.

Key words: Crowd dynamics models, crow dynamics simulators, parallel computing, performances, computational

INTRODUCTION

Over the last few decades, much attention has been devoted to pedestrian dynamics studies to provide solutions for problems such as massive congestion. Besides efforts on introducing better pedestrian facilities and to provide protection for these facilities (e.g., fire safety (Lo, 1999; Zhao *et al.*, 2004), continuous development of crowd dynamics models has also become a main agenda. Model-based simulators Pelechano and Badler, 2006; Silverman *et al.*, 2006) have become a desired tool for engineers, architects and emergency management and transportation agencies in examining the pedestrian facilities for pedestrian flow in both emergency and non-emergency situations.

Various key aspects are crucial for effective simulators. Basically, most simulators are based on mathematical models of pedestrian dynamic flow, therefore, establishing a good representative model is crucial to obtain more realistic performance of such simulators.

A pedestrian is considered a major mass during congestion. Understanding their behavior during interactions with other pedestrians is important as realistic aspect interactions can be considered or incorporated into the models when developing simulator tools. Consideration and subsequent implementation of the different aspects of evacuation in such tools is also essential to allow designers to achieve the desired evacuation results for different situations and physical environment layouts. Accordingly, for the evacuation process, modifying an evacuation model-based simulators have become the main goal for researchers in order to provide protection for pedestrian facilities (e.g., EGRESS (Ketchell *et al.*, 1993), EXODUS (Galea and Galparsoro, 1993; Thompson and Marchant, 1995), SGEM and FDS+EVAC (Korhonen *et al.*, 2010; Lo *et al.*, 2004).

One of the main challenges of simulator development is the optimization of computational time. Where it is expected that when the simulated mathematical model is complex and time consuming, then the corresponding simulator is also complex and time consuming. The main challenge for the simulator development is normally to decrease the model computational time. And this in turn depends the approaches used by the models where they are crucial in determining the computational efficiency of

Corresponding Author: Osama Moh'd Alia, Department of Computer Science,
Faculty of Computers and Information Technology, University of Tabuk, P.O. Box 741,71491 Tabuk, Kingdom of Saudi Arabia

the corresponding simulators. In addition to that, large-scale simulations of both normal and evacuation movements (Sarmady *et al*., 2011), especially involving 2D or 3D animations, demand high computational resources. This has led researchers to search for better solutions, where, among the ones considered is the use of parallel computing approaches. Parallel Computing (PC) basically entails the simultaneous execution of the same task on multiple processors in order to increase the processing power (Jordan and Alaghband, 2002). Due to this division of tasks into many sub-tasks performed by many processors, significant reduction in response time can be expected.

In this study we elaborate about PC-based approaches in the context of reducing computational demands for crowd dynamic simulators. This study also includes a comprehensive overview of state-of-the-art PC approaches as well as evaluations of the techniques employed in crowd dynamic simulators.

MATERIALS AND METHODS

Crowd dynamic models and computational efficiency: Two main classifications of Crowd Dynamics Models (CDM) are the macroscopic and microscopic models. Macroscopic models are more concerned with the macroscopic properties (variables) of a whole crowd such as its density (ρ), mean speed (V) and pedestrian flow (f) (Haight, 1963). These models are often based on traffic flow, queuing theory or fluid or continuum mechanics (Hughes, 2002). The latter models, namely microscopic models are more concerned with the detailed interactions between the pedestrians and their effect on motion. A variety of models have been proposed in microscopic studies, among them the cellular automata models, discrete choice models (Antonini *et al*., 2006; Robin *et al*., 2009) and the social force model (Helbing and Molnr, 1995; Helbing *et al*., 0000; Lakoba *et al*., 2005).

To determine the computational efficiency in the macroscopic models, close attention needs to be provided for the manipulation of the macroscopic variables. This is done by applying rules or processes for calculating the mentioned macroscopic variables which are to be assigned to each particle at each time-step in the corresponding simulators. Accordingly, in a macroscopic simulation, the estimation of steps number to be executed for calculating and assigning one variable (e.g. velocity) to N particles within one step of motion is K + N steps. K here is the number of steps required for calculating the rules or processes assigned to this variable. The microscopic models, on the other hand, assume the detailed interactions among the particles. For each particle, the mechanism representing the interactions requires its own computational processes. Therefore, based on microscopic simulation we have KN steps required to be executed while performing the same scenario mentioned above. In view of that, the computational processes are higher than what is required in the macroscopic models, whatever the number of simulated pedestrians.

Another important CDM classification is the nature of the basic variables used for the description of the particle system, namely space, time and state (e.g., velocity). Each of these variables can either be discrete or continuous (i.e., a real number). Although, the models with discrete variables are often an approximation (cellular automata models (Blue and Adler, 1999; Burstedde *et al*., 2001) and discrete choice models (Antonini *et al*., 2006; Robin *et al*., 2009), they have the advantage of being more time-efficient. For example, the physical environment in the Cellular Automata Models is usually considered as a floor area, divided into a lattice of cells of equal sizes. The cells from areas that can be occupied by obstacles, particles (pedestrians) or may remain as empty spaces. The size of the cell is governed by the chosen velocity, but limited by the size of the particle. Thereby, the particle moves from one cell to the adjacent cell, in one-step movements which should achieve the distance required for this movement. Hence, the value of the time step chosen for the simultaneous update process of the particle motion inside the environment is relatively high.

On the other hand, there are real situations that can only be reproduced using continuous variables. For example, a continuous space such as in the Social Force Model (SFM) generates more realistic pedestrian motion. Most equations of the forces of motion in this model require very small step-sizes in time (this by the way is required, else the simulation enters into an unstable regime (Lakoba *et al*., 2005) where it would falsely produce results showing invasive counterintuitive behaviors such as overlapping among particles). Therefore, the simulator has to resolve the motion of the particles on shorter time scales (which requires higher computational demand) as compared to discrete models. For comparison, a typical value for the time step used in most Cellular Automata Models is $t_{discrete} = 0.5$s whereas in the SFM, the time step value that is $>t_{cont} = 0.001$ s it would be most likely result in counterintuitive behaviors such as stated in (Lakoba *et al*., 2005). In principle and for efficiency examination we can consider the ratio $t_{discrete}/t_{cont.}$ of the average number of steps needed with discrete model versus the number of steps needed with continuous model for one step of motion. This is with making some oversimplification that the step of motion takes the same time to be executed in both models.

Implementations of the interactions among particles could be classified into two, namely rule-based and force-based approaches. Particle's motion in rule-based models are generated by decisions based on the interactions between the particles and their current situation (the surrounding particles and the physical environment) as well as their goals and so on. In force-based models, the interactions between particles are represented as forces responsible for their motion. The computation of the physical forces in the code mostly requires the implementation of numerical algorithms that unfortunately consumes more time than the implementation of decisions (the time required in performing if-statement, for example) in the rule-based models.

Another factor influencing computational efficiency is the incorporation of more mechanisms into the model to reproduce realism, such as the behavioral level of pedestrians. According to Hoogendoorn *et al.* (2002), pedestrian behavior can theoretically be divided into three interrelated levels. First is the strategic level where pedestrian activities and their order are determined. Second, the tactical level where decisions are made while pedestrians perform the activities (e.g., choosing a route (exit) to an intermediate target among alternative routes (exits) based on utility maximization). Third, the operational level where the instantaneous behaviors that involve most activities resulting from the interactions among pedestrians such as avoiding collisions, deviations, acceleration and deceleration are described. Daamen (2004) and Asano *et al.*, (2010) pointed out the importance of obtaining integrated models comprised of two complementary levels: the operational and the tactical levels. Accordingly, integrating a route/exit choice model (a form of intelligence on the part of the simulated pedestrians) into the operational-level models to grant the pedestrians of the existing microscopic model far-sighted decisions is an essential factor for obtaining more realistic models (Zainuddin and Shuaib, 2010a, b; Lo *et al.*, 2006; Ehtamo *et al.*, 2010; Huang and Guo, 2008; Zainuddin and Shuaib, 2011). However, such integrations and modifications for normal and evacuation situations increase the simulators computation processes. Reasonably, assume we have N processes performed in the operational level and M processes performed at the tactical level we have at least, due to the independence of these levels, N+M computational processes performed at each time step.

Parallel processing: This study gives an overview of parallel computing and their mechanisms. In addition to the techniques used to evaluate the performance of parallel computing methods.

Parallel mechanisms: Parallel computing is critical in many applications that require processing of large amounts of data. These are such as weather forecasting, data visualization, computational biology and engineering (Jaber *et al.*, 2014).

Parallel processing can conventionally have two classifications, namely implicit and explicit parallelism. Implicit parallelism is when the programming language can decide which parts of the task to run in parallel. No control scheduling of calculations is specified. The explicit parallelism on the other hand is when the programmer him/herself can allow or make certain parts of the code to execute in parallel.

Flynn (1966, 1972) proposed a different classification scheme that is based on instruction and data streams. He states that parallel processing can have a four-way classification which are SISD (Single Instruction Single Data), SIMD (Single Instruction Multiple Data), MISD (Multiple Instructions Single Data) and MIMD (Multiple Instructions Multiple Data).

Johnson (1988) proposed a new taxonomy-based memory structure, namely the shared/global memory and distributed memory (Fig. 1). Communications are totally involved between processors in parallel computation and the mechanism used for communication/synchronization is called message passing. Fortunately, many messages-passing libraries have been developed to provide routines to initiate and configure the messaging environment as well as to send/receive data packets between processors. The two most popular message-passing libraries are Parallel Virtual Machine (PVM) and Message Passing Interface (MPI) while the most popular routines as shared address space paradigms are the POSIX Thread and OpenMP.

Recently, researchers tried to use General Purpose computation on Graphics Processing Units (GPGPU) as a parallel programming approach. The GPGPU approach programs Graphics Processing Units (GPU) chips using an Application Programming Interface (API) functions such as Open GL, Direct3D and CUDA (Compute Unified Device Architecture) (Yong and Jun, 2010) in order to obtain faster results. However, GPUs are highly threaded streaming multiprocessors of very high computation and data throughput. Therefore in 2006, NVIDIA developed CUDA, a parallel computing platform and programming model implemented by the GPUs. CUDA has been widely d eployed in thousands of applications

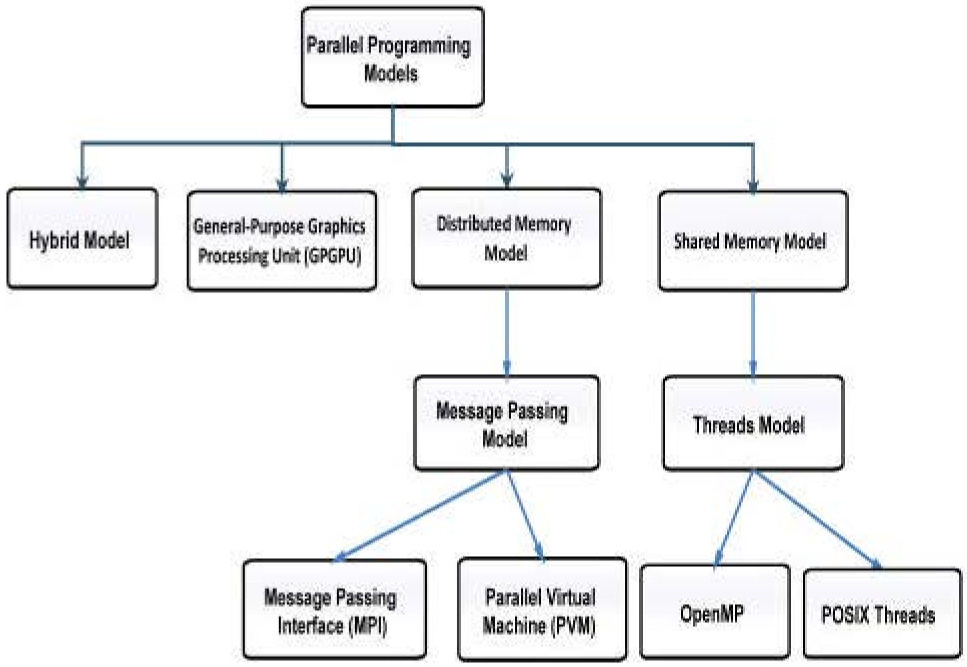
Parallel Programming Models
Hybrid Model
General-Purpose Graphics Processing Unit (GPGPU)
Distributed Memory Model
Shared Memory Model
Message Passing Model
Threads Model
Message Passing Interface (MPI)
Parallel Virtual Machine (PVM)
OpenMP
POSIX Threads

Fig. 1: Parallel programming models

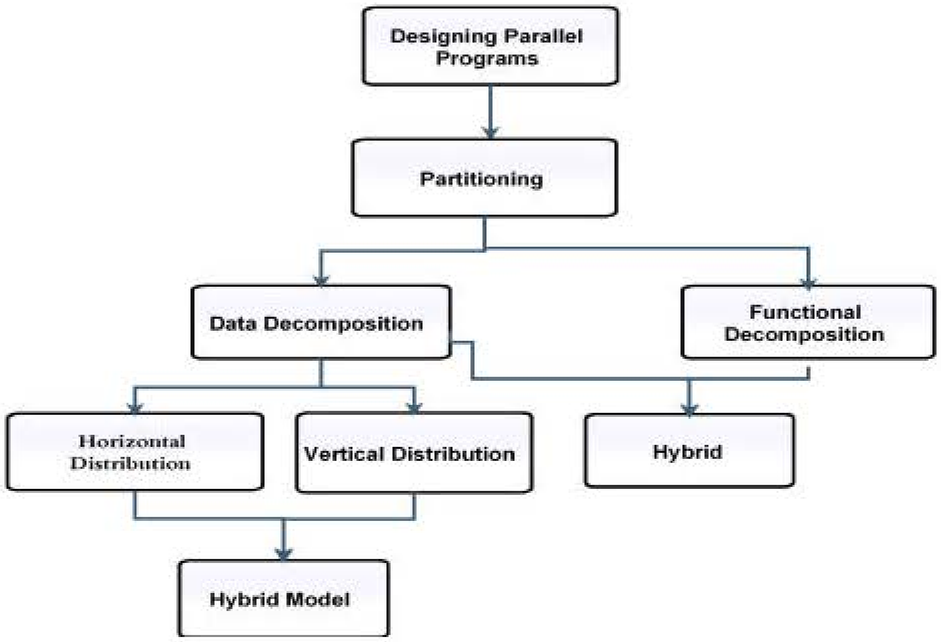

Fig. 2: Designing parallel programs

and published research studys such as astronomy, biology, chemistry, physics and data mining. Supported by an installed base of over 300 million CUDA-enabled GPUs in notebooks, workstations, compute clusters and supercomputers.

Two mechanisms worth mentioning to parallelize large amounts of data are function decomposition and data decomposition or both as shown in Fig. 2. Data decomposition involves the horizontal distribution or vertical distribution or both. Moreover, hybrid parallelism merges both data parallelism with horizontal or vertical distribution and function decomposition (Jaber *et al*., 2014).

RESULTS AND DISCUSSION

Parallel evaluation techniques: Many quantitative variables can be used to measure the parallel performance of a given application such as Amdahl's law, Speedup (S), Efficiency (E) and Overhead (T_o) (Jordan and Alaghband, 2002). The execution time (i.e., the serial runtime) of a program is the elapsed-time (T_s) between the beginning and the end of the program execution on a sequential machine. The parallel run time (T_p) is the elapsed-time from the moment a parallel computation starts to the moment the last processor finishes execution. Consequently, Overhead (T_o) or the total overhead of a parallel system, is the total collective time spent by all processing elements over and above that required by the fastest known sequential algorithm for solving the same problem on a single processing element (Eq. 1).

$$\text{Overhead}(T_o) = p \times T_p - T_s \quad (1)$$

Speedup (S) captures the relative benefit of solving a problem in parallel. It is defined as the ratio of the time taken to solve a problem on a single processing element T_s to the time required to solve the same problem on a parallel computer T_p with identical processing elements p; given by Eq. 2:

$$\text{Speedup}(S) = T_s/T_p \quad (2)$$

The Efficiency (E) is a measure of the fraction of time for which a processing element is usefully employed; it is defined as the ratio of speedup S as calculated in Eq. 2 to the number of processing elements p. The formula is shown in Eq. 3:

$$\text{Efficiency (E)} = S/p \quad (3)$$

Parallel crowd dynamics models: Different parallel methods were introduced in order to address issues in computational complexities of crowd dynamic simulations. Substantial efforts were focused on using:

- Multicore programming based on multithreading
- GPGPU
- MPI libraries

In the remaining parts of this section, existing parallel methods for crowd dynamic simulation are discussed, with a summary given in Table 1.

Table 1: Summary of the state-of-the-art parallel computing approaches used in crowd dynamic simulators

Authors (year)	Parallelism data function	Shared/ distributed memory	Parallelism model	Serial complexity	Parallel complexity	Dataset
Clayton *et al* (2009)	Data parallelism	Shared/global	Multithreading	-	-	200 agents
Vigueras *et al*. (2008a)	memory		-	-	23000 agents	
Vigueras *et al*. (2008b)			-	-	23000 agents	
Vigueras				-	-	8000 agents
Vigueras *et al*. (2008)				-	-	8000 agents
Lozano *et al*. (2007)				-	-	8000 agents
Stephen				-	3.8X speedup	5000 and 25000agents
Richmond and Romano	Data parallelism	Shared/global memory	GPU	$O(n^2)$	-	4000 and 32000 agents
			$O(n^2)$	-	4000 and 32000 agents	
Joselli *et al*. (2009)				$O(n^2)$	-	1 thousand to 1 million boids
MINTL (2012)				CPU 100000 ms	GPU 1000 ms	2000×2000 distance matrix
MINTL (2013)				-	-	Town square and building
Ugo *et al*. (2009)				$O(n^2)$	0.25X	8192 and 65536 individuals
Lysenko and Souza (2008)				O(nlogn)	-	16 million concurrent agents
Erdal *et al*. (2009)				6,282.52 ms	61.20-ms	one million virtual people
Alessandro				$O(n^2)$	-	1024-65536 Boids
Quinn *et al*. (2003)	Data parallelism	Distributed	MPI	1 worker 5 updates	10 worker 50 updates/sec	Airport eva. 10,000
Pedestrians						
Bo and Suiping		memory		$O(n^2)$	Proc = 1,t = 25.24, Proc = 16, t = 3.36	512 Boids
Porte				3 works, 61 cycles/sec 16 works, 1.2 cycles/sec	10,000 Pedestrian	
Solar *et al*. (2010)				-	-	80000, 320000 individuals
Solar *et al*. (2011)				$O(n^2)$	O (nm)	524288, 1048576 individuals
Solar *et al*. (2012)				$O(n^2)$	O (nm)	262144, 524288 individuals
Solar *et al*. (2013)				-	Speedup 12.39, 16.30 and 21.60	131.072, 262.144 and 24.288 5 individuals
Armstrong *et al*. (2008)				-	0.25X speedup	2 million agents

Multicore programming based on multithreading: Clayton *et al*. (2009) implemented a multi agent-based design pattern developed by the CoSMoS research group at Edinburgh Napier University, Merchiston Campus. Given the nature of Multi Agent Systems (MAS), parallel processing techniques were used in their implementation. The CoSMoS design patterns are based on Communicating Sequential Processes (CSP). Processes have their own thread of control and entirely encapsulate their data and maintain their own state. However, this approach did not clearly explain the parallel technique used and the experiments were performed when the number of agents was increased. The researchers also did not mention the changing number of threads, number of processors and their time. Evolutions were also not performed using standard measurements such as speedup, overhead and etc.

Vigueras *et al*. (2008a, b) proposed an architecture for parallelizing the action server for crowd simulation and the distribution of the semantic database. The action server parallelization technique was based on agent-based modeling. However, the researchers used multithreading to parallelize the action server where the action server was divided into a set of processes so that each process is executed in parallel on different computers to solve system bottleneck issues by Vigueras *et al*. (2008). Vigueras *at el*. (2008) demonstrated a partitioning method for distributed crowds using irregular shape regions. This approach finds the near optimal partition of regions of agents that minimizes the number of agents near the borders of the regions. However, this researcher did not mention clearly the parallel technique that was used.

Guy *at el*. (2009) presented a parallel collision avoidance approach for multi agent real-time simulation called ClearPath, where they used data decomposition for parallelization. However, multicore programming based on multithreading was used to implement the algorithm.

General-Purpose Computation on Graphics Processing Units (GPGPU): Richmond and Romano proposed an agent-based pedestrian dynamics model on the GPU. However, the researchers used the OpenGL programming to mapping the agent data.

Joselli *at el*. (2009) proposed a 2D and 3D data structure which they termed neighborhood grid. This was used for massive crowd simulation on a GPU using CUDA programming. In this data structure, each cell fits only one entity of position. However, the CUDA thread is executed in parallel for each entity. The researchers used the data decomposition techniques to divide the task among GPU threads.

Mintl (2012) presented the distance matrix calculation for modelling pedestrian movement implemented using the GPU. Two pedestrian simulation models were combined, namely the cellular automata and social force model. Data decomposition was used to divide the distance matrix into cells. Cells can be calculated in parallel in one step. Another work by the same research (Mintal, 2014) implemented a multi-agent simulation architecture on GPU using OpenGL.

Presented a GPU implementation of an individual based simulation model for fish schools and animal groups based on CUDA programming. Their researcher allows the simulation of the collective motion of high-density individual groups. The simulation uses grid cells to keep track of the individuals' positions and offloads the sorting to build up the data grid structure to the GPU. The sorting is performed inside the cells to optimize the search and then to quickly obtain information about neighbors for each individual, in parallel.

Lysenko and Souza (2008) presented data parallel algorithms for simulating agent-based models. These include methods for handling environment updates and agent interactions on the GPU, Erra *at el*. (2009) proposed the GPU implementation of crowd simulation for virtual marathons using Visual C++ and CUDA programming. Their experiments involved more than one million virtual people (32,768 runners and 1,015,808 spectators).

Silva *at el*. (2009) implemented (on the GPU) Reynolds Boids model for simulating large groups. They tested their implementation using two GPU programming languages: one using the Cg shader language and the other using CUDA. The implementation uses global memory to store several arrays representing Boids information.

Message Passing Interface (MPI): Quinn implemented the parallel SFM using the C programming language with the MPI library. The researchers used the task/farm parallelism model to manage the processors. This parallel implementation of the SFM used eleven processors to simulate 10,000 evacuating pedestrians on a cluster connected by a gigabit switch.

Zhou and Zhou (2004) presented a parallel algorithm to simulate the flocking behavior of a large group using MPI. They used the data decomposition mechanism to partition the space. Dynamic load balancing scheme is used to manage the partitioning among processors.

Porte and Thalmann (2005) proposed a real time simulation of a large crowd of 10,000 pedestrians on a cluster of machines using the MPI mechanism. The Workload-balancing algorithm used in their implementation allowed best performance to be kept during the simulation.

Solar *at el.* (2010), presented a proximity load balancing approach for a distributed cluster-based individual-oriented fish school simulator. The researchers implemented this algorithm using MPI with varying number of processors, i.e., 1, 4, 8, 16, 32 and 64.

Armstrong *at el.* (2008) developed the Parallel Particle Data Model (PPDM) for Agent-based modelling and discrete event simulation by Message Passing Interface (MPI) for Java.

Analysis and comparison: In this study we compare the performances of the state-of-the-art parallel computing approaches used in crowd dynamic simulators. A summary is given in Table 1. Two evaluations are performed, specifically relating to execution time and computational complexity. Execution time refers to when a program is running (executing) whereas computational complexity (or also called the abstraction level) is the count of the number of instructions or statements executed using a mathematical notation Big-O notation. However, it is difficult to compare the algorithm execution time using a real time measurement approach due to different architectures, hardware and speed. Execution times are specific to a particular computer because different computers run at different speeds. Moreover, most crowd dynamics models' source codes are not publicly available to researchers. This is where the computational complexity comes in where it is used to compare these algorithms. The size and type of dataset used in these algorithms were also introduced in this comparison.

From Table 1, it can be noticed that the complexity of the most sequential simulations is O(n^2) such as Joselli *et al.* (2009), Solar *et al.* (2011) while the case in (Lysenko and Souza, 2008) is O(nlogn). Some of these simulators mention their complexity improvement while most do not. Among works that mention improvement are Solar *et al.* (2011), Solar *et al.* (2012) where the simulation complexity improved form O(nm) to O(n^2) as shown in Table 1. By using 64-cores with workloads of 131072, 262144 and 523288 individuals, Solar *at el.* (2013) reported that the speedup of their proposed parallel model were 12.39X, 16.30X and 21.60X, respectively. On the other hand, some researchers mentioned improvement gained from implementing the parallel techniques in terms of time speed such as. When they simulate 512 boids on one processor, the total required time is 25.24 ms whereas using 16 processors required 3.36 ms. Porte and Thalmann (2005), the researcher simulated 10,000 pedestrians in 225 areas using 3 and 16 workers where the results were 61 and 1.2 cycles/sec, simultaneously. Erra *et al.* (2009), the researchers simulated a virtual marathon with one million virtual people (32768 runners and 1015808 spectators) in 6,282.52 ms. They managed to improve that time to a mere 62.20 ms with 100X speedup on a GPU using CUDA. Guy *et al.* (2009) mentioned that the P-ClearPath achieves around 3.8X parallel speedup on the quad-core using 5000 and 25000 agents. MINTL (2013) simulated a 2000×2000 distance matrix where the processing time using CPU was 100000-ms while GPU was 1000-ms. The researchers in mentioned that the GPU-based implementation is 0.25X better than the ABGPU-based implementation proposed by. The researchers in reported that with 1 worker (i.e. processor) the system is able to update the positions of the 10,000 simulated pedestrians nearly 5 updates per second while when they used 10 workers (11 processors overall), the system is able to update these positions with nearly 50 updates per second. Armstrong *at el.* (2008) achieved fairly good scaling with the 64 nodes case within twenty-five percent speedup of the ideal result.

For the rest, the researchers used charts to compare their results. Therefore, extracting execution time, speedup and other evaluation techniques are impossible.

Regarding the type of the dataset used in the parallel computing-based crowd dynamic simulators, actually, different types were used such as human agents movement, fish schooling, flocking simulation, virtual marathon, airport evacuation, etc. The size of the dataset varies from small to medium to large and very large. Clayton *et al.* (2009), the researchers used a very small dataset consisting of 200 agents and the average time per agent tends to settle at around 0.78 ms. Also, in the researchers simulate flocks with 512 Boids. The researchers by Vigueras *et al.* (2008), simulated a small dataset consisting of 8000 agents while (Yilmaz *et al.*, 2009) simulated 23000 agents. Researchers in and simulated 4000 and 32000 agents. Simulated a school of fish with about 8192 and 65536 individuals. Silva *et al.* (2009) simulated Reynolds Boids model that had a range of boids from 1024-65536. Mintal (2014) simulated town square and building floor (pedestrian movement simulation). An airport evacuation simulation with 10,000 pedestrians were studied by Porte and Thalmann (2005) used 10,000 pedestrians in their simulation. Solar *at el.* (2010) used a fish school model in their simulation with a large dataset consisting of 80,000 fishes and 320,000 fishes while the same researchers in (Solar *et al.*, 2011) increasedthe dataset to 524,288 and 1,048, 576 individuals. The same researchers by Solar *et al.* (2013) used 131072, 262144 and 524288 individuals. A very large dataset was simulated in Joselli *et al.* (2009), Lysenko and Souza, (2008) and Erra *et al.* (2009). Joselli *et al.* (2009) simulated flocking boids with a very large dataset ranged from 1 thousand to 1 million boids. Lysenko and Souza (2008) simulated SugarScape and StupidModel (benchmark for agent based modeling toolkits) with 16 million concurrent agents on grid sizes of up to 4096×4096. The Virtual Marathon with one million virtual people (32,768 runners and 1,015,808 spectators) was simulated by Erra *et al.* (2009). Armstrong *et al.* (2008) simulated 2 million agents on 16 processors.

CONCLUSION

In this review, we underlined the origin sources of the problem of the high computational demands of the crowd dynamic models and their effects on the performance of their simulators. In addition to that we also underlined research efforts at solving this problem, specifically to use parallel computing approaches. We also conducted a review of the many published state-of-the-art approaches which hopes to form the direction of future research in the area. We also hope that solutions can be offered based on this review, to the problem using multiple methods of parallel computing approaches in order to reduce complexity, in both time and memory storage requirements. It can be concluded from this study that, in spite of some limitations in the existing parallel-based crowd dynamic models presented in this study, the parallel processing methods are able to provide practical solutions. The need for further research in utilizing parallel computing methods to solve this problem is still much desired. Future efforts can possibly be directed to the implementation of parallel techniques such hybridization between distributed and shared memory models, hybridization between GPU and CPU methods or hybridization between data decomposition and function decomposition.

ACKNOWLEDGEMENT

Researchers would like to thank the Deanship of Scientific Research, University of Tabuk for their financial support (Grant no. S183-1434).

REFERENCES

Antonini, G., M. Bierlaire and M. Weber, 2006. Discrete choice models of pedestrian walking behavior. Transport.Res. Part B: Methodol., 40: 667-687.

Armstrong, R., B. Allan, M. Goldsby, Z. Heath and M. Shneider *et al.*, 2008. Parallel computing in enterprise modeling. Sandia National Laboratories, Livermore, California. http://s3.amazonaws.com/academia.edu.documents/43531331/Parallel_Computing_in_Enterprise_Modelin20160308-23401-17wkfxh.pdf?AWSAccessKeyId=AKIAJ56TQJRTWSMTNPEA&Expires=1484198465&Signature=olXr2lsIL%2Fyx

Asano, M., T. Iryo and M. Kuwahara, 2010. Microscopic pedestrian simulation model combined with a tactical model for route choice behaviour. Trans. Res. Part C: Emerging Technol., 18: 842-855.

Blue, V. and J. Adler, 1999. Cellular automata microsimulation of bidirectional pedestrian flows. Transp. Res. Record J. Transp. Res. Board, 1678: 135-141.

Burstedde, C., K. Klauck, A. Schadschneider and J. Zittartz, 2001. Simulation of pedestrian dynamics using a two-dimensional cellular automaton. Physica A: Stat. Mech. Appl., 295: 507-525.

Clayton, S., N. Urquhard and J. Kerridge, 2009. Application of CoSMoS parallel design patterns to a pedestrian simulation. Parallel Processing and Applied Mathematics, Wyrzykowski, R., J. Dongarra, K. Karczewski and J. Wasniewski (Eds.). Springer, Berlin, Germany, ISBN:978-3-642-14402-8, pp: 505-512.

Daamen, W., 2004. Modelling passenger flows in public transport facilities. Ph.D Thesis, TU Delft, Delft University of Technology, Delft, Netherlands.

Ehtamo, H., S. Heliovaara, S. Hostikka and T. Korhonen, 2010. Modeling Evacuees' Exit Selection with Best Response Dynamics. In: Pedestrian and Evacuation Dynamics 2008, Klingsch, W.W.F., C. Rogsch, A. Schadschneider and M. Schreckenberg (Eds.). Springer, Berlin, Germany, ISBN:978-3-642-04503-5, pp: 309-319.

Erra, U., B. Frola, V. Scarano and I. Couzin, 2009. An efficient GPU implementation for large scale individual-based simulation of collective behavior. Proceeding of the 2009 HIBI'09 International Workshop on High Performance Computational Systems Biology, October 14-16, 2009, IEEE, Salerno, Italy, ISBN:978-0-7695-3809-9, pp: 51-58.

Flynn, M.J., 1966. Very high-speed computing systems. Proc. IEEE., 54: 1901-1909.

Flynn, M.J., 1972. Some computer organizations and their effectiveness. IEEE. Trans. Comput., 100: 948-960.

Galea, E.R. and J.P. Galparsoro, 1993. Exodus: An evacuation model for mass transport vehicles. Civil Aviation Authority, 22: 341-366.

Guy, S.J., J. Chhugani, C. Kim, N. Satish and M. Lin *et al*., 2009. Clearpath: Highly parallel collision avoidance for multi-agent simulation. Proceedings of the 2009 ACM SIGGRAPH-Eurographics Symposium on Computer Animation, August 01-02, 2009, ACM, New York, USA., ISBN:978-1-60558-610-6, pp: 177-187.

Haight, F.A., 1963. Mathematical Theories of Traffic Flow. Academic Press, New York, USA.,.

Helbing, D., I. Farkas and T. Vicsek, 2000. Simulating dynamical features of escape panic. Nature, 407: 487-490.

Hoogendoorn, S.P., P.H. Bovy and W. Daamen, 2002. Microscopic pedestrian wayfinding and dynamics modelling. Pedestrian Evacuation Dyn., 2002: 123-154.

Huang, H.J. and R.Y. Guo, 2008. Static floor field and exit choice for pedestrian evacuation in rooms with internal obstacles and multiple exits. Phys. Rev. E., Vol. 78.

Hughes, R.L., 2002. A continuum theory for the flow of pedestrians. Transport. Res. Part B: Methodol., 36: 507-535.

Jaber, K.M., R. Abdullah and N.A.A. Rashid, 2014. Fast decision tree-based method to index large DNA-protein sequence databases using hybrid distributed-shared memory programming model. Int. J. Bioinf. Res. Applic., 10: 321-340.

Johnson, E.E., 1988. Completing an MIMD multiprocessor taxonomy. ACM. Sigarch Comput. Archit. News, 16: 44-47.

Jordan, L.E. and G. Alaghband, 2002. Fundamentals of Parallel Processing. Prentice Hall, Upper Saddle River, New Jersey, ISBN:0139011587,.

Joselli, M., E.B. Passos, M. Zamith, E. Clua and A. Montenegro *et al*., 2009. A neighborhood grid data structure for massive 3d crowd simulation on GPU. Proceeding of the 2009 VIII Brazilian Symposium on Games and Digital Entertainment, October 8-10, 2009, IEEE, Niteroi, Brazil, ISBN:978-1-4244-6011-3, pp: 121-131.

Ketchell, N., S. Cole, D.M. Webber, C.A. Marriott and P.J. Stephens *et al*., 1993. The EGRESS Code for Human Movement and Behaviour in Emergency Evacuations. In: Engineering for Crowd Safety, Smith, R.A. and J.F. Dickie (Eds.). University of Edinburgh, Edinburgh, Scotland, pp: 361-370.

Korhonen, T., S. Hostikka, S. Heliovaara and H. Ehtamo, 2010. FDS+ Evac: An Agent Based Fire Evacuation Model. In: Pedestrian and Evacuation Dynamics 2008, Klingsch, W.W.F., C. Rogsch, A. Schadschneider and M. Schreckenberg (Eds.). Springer, Berlin, Germany, ISBN:978-3-642-04503-5, pp: 109-120.

Lakoba, T.I., D.J. Kaup and N.M. Finkelstein, 2005. Modifications of the helbing-molnar-farkas-vicsek social force model for pedestrian evolution. Simulation, 81: 339-352.

Leopold, C., 2001. Parallel and Distributed Computing: A Survey of Models, Paradigms and Approaches. John Wiley & Sons, New York, USA., ISBN:0471358312,.

Lo, S.M., 1999. A fire safety assessment system for existing buildings. Fire Technol., 35: 131-152.

Lo, S.M., Z. Fang, P. Lin and G.S. Zhi, 2004. An evacuation model: The SGEM package. Fire Saf. J., 39: 169-190.

Lozano, M., P. Morillo, J.M. Orduna and V. Cavero, 2007. On the design of an efficient architecture for supporting large crowds of autonomous agents. Proceeding of the 21st International Conference on Advanced Information Networking and Applications (AINA'07), May 21-23, 2007, IEEE, Valencia, Spain, ISBN: 0-7695-2846-5, pp: 716-723.

Lysenko, M. and D.R.M. Souza, 2008. A framework for megascale agent based model simulations on graphics processing units. J. Artif. Societies Soc. Simul., 11: 10-27.

Mintal, M., 2012. Accelerating distance matrix calculations utilizing GPU. J. Inf. Control Manage. Syst., 10: 71-80.

Mintal, M., 2014. Framework for utilizing computational devices within simulation. Acta Inf. Pragensia, 2: 59-67.

Pelechano, N. and N.I. Badler, 2006. Improving the realism of agent movement for high density crowd simulation. Ph.D Thesis, Center for Human Modeling and Simulation, University of Pennsylvania, Philadelphia, Pennsylvania.

Porte, P. and D. Thalmann, 2005. Real-Time Simulation of a Large Crowd on a Cluster of Machines. VRLab Publisher, San Francisco, California,.

Robin, T., G. Antonini, M. Bierlaire and J. Cruz, 2009. Specification, estimation and validation of a pedestrian walking behavior model. Transp. Res. Part B Methodol., 43: 36-56.

Sarmady, S., F. Haron and A.Z. Talib, 2011. A cellular automata model for circular movements of pedestrians during Tawaf. Simul. Modell. Pract. Theory, 19: 969-985.

Silva, A.R.D., W.S. Lages and L. Chaimowicz, 2009. Boids that see: Using self-occlusion for simulating large groups on gpus. Comput. Entertainment CIE., 7: 1-20.

Silverman, B.G., M. Johns, J. Cornwell and O.K. Brien, 2006. Human behavior models for agents in simulators and games: Part I: Enabling science with PMFserv. Presence Teleoperators Virtual Environ., 15: 139-162.

Solar, R., F. Borges, R. Suppi and E. Luque, 2013. Improving communication patterns for distributed cluster-based individual-oriented fish school simulations. Procedia Comput. Sci., 18: 702-711.

Solar, R., R. Suppi and E. Luque, 2010. High performance individual-oriented simulation using complex models. Procedia Comput. Sci., 1: 447-456.

Solar, R., R. Suppi and E. Luque, 2011. High performance distributed cluster-based individual-oriented fish school simulation. Procedia Comput. Sci., 4: 76-85.

Solar, R., R. Suppi and E. Luque, 2012. Proximity load balancing for distributed cluster-based individual-oriented fish school simulations. Procedia Comput. Sci., 9: 328-337.

Thompson, P.A. and E.W. Marchant, 1995. Computer and fluid modeling of evacuation. Saf. Sci., 18: 277-289.

Vigueras, G., J.M. Orduna and M. Lozano, 2010. Performance improvements of real-time crowd simulations. Proceeding of the 2010 IEEE International Symposium on Parallel and Distributed Processing, Workshops and Phd Forum (IPDPSW), April 19-23, 2010, IEEE, Valencia, Spain, ISBN: 978-1-4244-6534-7, pp: 1-4.

Vigueras, G., M. Lozano, J.M. Orduna and F. Grimaldo, 2008a. Improving the performance of partitioning methods for crowd simulations. Proceeding of the HIS'08 8th International Conference on Hybrid Intelligent Systems, September 10-12, 2008, IEEE, Valencia, Spain, ISBN:978-0-7695-3326-1, pp: 102-107.

Vigueras, G., M. Lozano, C. Perez and J.M. Orduna, 2008b. A scalable architecture for crowd simulation: Implementing a parallel action server. Proceeding of the 2008 37th International Conference on Parallel Processing, September 9-12, 2008, IEEE, Valencia, Spain, ISBN:978-0-7695-3374-2, pp: 430-437.

Yilmaz, E., V. Isler and Y.Y. Cetin, 2009. The virtual marathon: Parallel computing supports crowd simulations. IEEE. Comput. Graphics Appl., 29: 26-33.

Yong, R.G. and H.H. Jun, 2010. Logit-based exit choice model of evacuation in rooms with internal obstacles and multiple exits. Chin. Phys. B., Vol. 19,

Zainuddin, Z. and M. Shuaib, 2010a. Modification of the decision-making capability in the social force model for the evacuation process. Transp. Theory Statist. Phys., 39: 47-70.

Zainuddin, Z. and M. Shuaib, 2010b. Incorporating decision making capability into the social force model in unidirectional flow. Res. J. Applied Sci., 5: 388-393.

Zainuddin, Z. and M.M.A. Shuaib, 2011. Modelling the independence factor and its effect on the preferred force of the social force model in emergency and non-emergency situations. Appl. Math. Inf. Sci., 5: 53-64.

Zhao, C.M., S.M. Lo, J.A. Lu and Z. Fang, 2004. A simulation approach for ranking of fire safety attributes of existing buildings. Fire Saf. J., 39: 557-579.

Zhou, B. and S. Zhou, 2004. Parallel simulation of group behaviors. Proceedings of the 36th Conference on Winter Simulation, December 05-08, 2004, ACM, Washington, DC., ISBN:0-7803-8786-4, pp: 364-370.