Parallelization of Sequential Monte Carlo methods in building occupancy simulation and data assimilation

Dr. Sanish Rai, West Virginia University Institute of Technology

SANISH RAI is an Assistant Professor in the Department of Computer Science and Information Systems at West Virginia University Institute of Technology, Beckley, WV. He received his Ph.D. degree from Georgia State University in 2017. His research interests include simulation and modeling, agent and graph based systems, artificial intelligence, data assimilation and machine learning. His email address is sanish.rai@mail.wvu.edu

Parallelization of Sequential Monte Carlo methods in building occupancy simulation and data assimilation

Sanish Rai
Department of Computer Science and Information Systems
West Virginia University Institute of Technology
410 Neville St., Beckley, WV, USA
sanish.rai@mail.wvu.edu

ABSTRACT

Building occupancy models for large building and occupancy is a computationally costly simulation. Data can be collected about the building from various sensors installed in the building and can be used for real-time estimation modelling. The simulation model and real-time estimation algorithm both are complex models requiring resource to accommodate for increase in size of the building environment, its occupants and their activities. In our previous work, we presented a graph-based agent-oriented model to efficiently simulate large number of occupants and used Sequential Monte Carlo (SMC) methods to assimilate the occupancy data for real-time estimation. A main issue with SMC algorithm is its high computational cost, which increases proportionally with the simulation model size. To manage the need of resource when increasing particles, we implement parallelization to SMC and improve the runtime of overall building occupancy simulation and estimation.

INTRODUCTION

Building occupancy simulation models the dynamic spatial-temporal behavior and activities of occupants in buildings. Studying and knowing the occupancy dynamics of a building is useful in multiple applications, including an effective evacuation of buildings with many occupants, energy conservation in buildings based on occupancy presence, design and monitor of smart environments, real-time crowd management in high-density places such as airports and train stations, and so on. Various simulation models have been developed to study the dynamics of building occupancy. These models simulate the occupancy patterns, occupancy behavior, and their interactions with each other as well as with the environment. Meanwhile, advances in sensor technology allow more and more buildings to be equipped with sensors, providing real-time information about the environment. Assimilating such sensor data into a simulation model allows better state estimation of building occupancy and can lead to more accurate simulation results. This process of integrating observation data from the real world into a simulation model to produce better estimates of system states is known as data assimilation. For data assimilation, both a simulation model that captures the state transition of the system and a method that assimilates real-time data into the model is needed.

In literature, we can find many models for building occupancy simulation, out of which agent-based and graph-based are two widely used. Agent-based simulation considers each occupant as an agent in a given environment, and the dynamic process of occupants is simulated over time to generate the complex and intriguing emergent behavior [1]. Graph-based models use the graph (node and edges) to represent the building structure and model the occupancy dynamics using some flow or queuing network [2]. While agent-based simulation has the advantage of being able to represent each occupant's behavior and decision making in detail, computation cost increases proportionally with the number of agents and their agents making it difficult to simulate large occupancy areas such as a game stadium, airports, rail terminals, etc.

On the other hand, a graph-based model assumes occupants as a homogeneous mass and models their flow across the graph structure. Although it can model large numbers of occupants efficiently, it fails to model the unique individual behavior of the occupants thus lacking emergent behaviors due to agent interactions.

To address the limitations of the models and take advantage of the good features of both models, we developed a new model called graph-based agent-oriented model [3] that combines the better property of both models, i.e. scalability and diversity. In this model, the graph-based feature means occupants' movement is modeled as flows from one node to another through a graph representing the building environment, and the agent-oriented feature refers to the model's capacity to consider agents as individual entities with unique features and decision-making capabilities. The model has the advantage that it can capture the heterogeneity of individual occupants while in the meantime can efficiently simulate thousands of occupants. This advantage is critical for supporting data assimilation for buildings with a large number of people like the game stadium and shopping malls.

While the graph-based agent-oriented model was able to model behaviors of building occupants, it lacks the analytic structures like those in equation-based numerical models. This makes it difficult to apply conventional estimation techniques such as Kalman filter and its variations for data assimilation. On the other hand, SMC methods are non-parametric filters with effective state estimation methods for systems with nonlinear non-Gaussian behavior. Thus, we choose SMC methods to carry out data assimilation in our work. In SMC methods, the system state underestimation is represented by a large set of random samples, named particles, and their associated weights. These particles are propagated over time using importance sampling and resampling mechanisms.

Sequential Monte Carlo (SMC) methods, also known as Particle Filters, are a set of sample-based methods that recursively estimate the state of a dynamic system from observation data using Bayesian inference and stochastic sampling techniques [6]. SMC methods represent the probability density function as a set of samples, each of which is known as a particle, with an associated weight. Different methods like perfect sampling, sequential importance sampling and resampling, acceptance-rejection sampling, and others are used to generate samples for the particles. Particle filters have an advantage of being able to represent arbitrary probability densities without requiring all information about the structure of the system model, making it an effective method for supporting data assimilation with sophisticated simulation models. At the same time, particle filters are iterative methods that can recursively adjust their estimations of system states when new observation data becomes available. This feature is desirable for systems where new sensor data arrive sequentially and the simulation system is continuously updated.

While SMC methods enable data assimilation for building occupancy simulation using the graph-based agent-oriented model, several challenges exist for using the models efficiently. The high dimensional system states cause a major challenge. As the size of the building structure increases, the number of nodes consisting of occupants in the graph model increases. This increases the dimension of the system state, and thus require more particles in order to estimate the system states accurately. Also, the complexity of the particle representing the system state increases. These particles consume a lot of computational resources, as such the overall run time of the simulation increases causing issues with resource utilization in the hardware.

One convenient way to address the issue is to utilize the hardware resource efficiently. With generous availability of multiprocessors and abundance of libraries to conveniently parallelize programs, it will be a missed opportunity if we do not take advantage of parallel computing before researching on algorithms for making SMC algorithms efficient. In this work, we focus mainly on parallelizing the sampling step of the SMC algorithm with easily available Java libraries for parallel programming. We conveniently convert our

simulation and data assimilation framework from serial to parallel and improve the runtime of the overall simulation. In the following sections we discuss the related works, briefly introduce the graph-based agent-oriented and the data assimilation model, discuss how to parallelize the SMC algorithm, present the experiment results and discuss the results. A conclusion is presented at the end to summarize the work.

RELATED WORKS

Building occupancy simulation is concerned with modeling and analysis of occupant property and behaviors. Authors in [4] have defined occupancy at four levels varying with time: the number of occupants in buildings, the occupancy status of space, the number of occupants in space and, the spatial location of occupants. Building occupancy finds its wide use in conserving resources such as the author in [5] have developed a framework known as sensor utility network which uses information from various sensors to dynamically estimate the occupancy and conserve resources. Building occupancy provides a basis for a smart environment where we can simulate our test beds with various sensors and actuators. In their work [6], the authors discuss the latest research in smart environment, philosophical and computational architecture considerations, network protocols, intelligent sensor networks and powerline control of devices, and action prediction and identification for the smart environment.

The data assimilation algorithm used in this research is SMC methods which is also known as Particle Filters (PF). A framework of dynamic data-driven simulation based on PF is presented by [7] for the forest fire spread simulation. In another research [8], PF is used to develop a framework and algorithms to solve the problems of positioning, navigation, and tracking. The author presents a general algorithm based on marginalization enabling a Kalman filter to estimate all position derivatives. PF finds use in other fields like biology and chemistry as well. The authors in [9] have presented a data assimilation framework for estimating movement patterns of about six occupants in an office environment. In one of our previous work [10], we developed a framework for using data assimilation for binary sensor data to predict occupancy behaviors in smart environments.

Researchers have implemented the SMC algorithm in various applications with parallelization of different components in the main algorithm. In [11] the author argues that the Monte Carlo algorithms are ideal suitable for parallel computing and Monte Carlo methods should be widely used parallel rather than sequential. The author mentions that a computer processor becomes cheaper and more plentiful, parallel computing will be prominent and Monte Carlo methods can also benefit from taking advantage of parallelization. In [12] authors used OpenMP in Intel C++ Compiler as well as with NVIDIA K20 GPU using CUDA 5.0 to parallelize over CPU and GPU. Their work contributes two new algorithms using Metropolis resampling and Reject resampling to make parallelization of the cumulative sum over weights process easier. In another work [13], the authors present a framework for parallel SMC-Probability Hypothesis Density filtering based on multiple processors that shows considerable parallelization speedup. All the steps of the algorithm are parallelized and while the state prediction step is parallelized in terms of distributing particles; the weight updating, resampling and estimate extraction steps are parallelized in terms of distributing measurements.

GRAPH-BASED AGENT-ORIENTED MODEL AND THE PF-BASED DATA ASSIMILATION FRAMEWORK

The graph-based agent-oriented model is a low-resolution simulation model which describes the dynamics of the agents at a higher abstraction level. In the model, the structure of the building is represented using nodes and edges, and occupants are represented as agents with basic properties required for the movement. The movement of the occupants is modeled as flow across the nodes through the links using queuing and

flow theory. Since movements of occupants in the network are more orderly organized, occupant behaviors at higher resolution are not necessarily required for the simulation. Lower resolution models like graph models can well represent the occupancy behaviors in situations like crowd and congestion. This significantly reduces the computational complexity of the simulation without losing the overall property of the occupancy behavior. Figure 1(a) represents the framework for a graph based agent-oriented model which show the agents and graph model. The graph model consists of the queues, and nodes and links manage the building structure. The agents are provided with decision-making capabilities which are governed by the rule engine maintained by the user. We also provide a GUI based interface where the occupant's behavior can be observed in a building environment.

More specifically, the building environment is represented using vertex for nodes and connecting structures like doors, stairs by links. The occupant in the model is represented using a variable *o* having state *s*, where

$$o =< ph, v_{position}, id, g_{id}, l_{from}, s >$$

$$s = \begin{cases} staying \\ moving \\ inqueue \end{cases}$$

Here, *ph* represents the phase of the occupant which represents the environment condition of the building, *vposition* represents the current vertex the occupant resides in, *id* represents the identification of the occupant, l_{from} represents the link that the occupant comes from, $_s$ represents the state of the occupants and has three different values, *staying*, *moving* and *inqueue*. The occupant has three basic states, staying to represent that the occupant is staying in a node, moving to represent that the occupant is moving across the node and inqueue to represent that the occupant is waiting to enter a node. To enter a node, the occupants follow the basic queue behavior. Each link has a flow rate and occupants enter based on the flow rate. If the node is full or the flow rate is low due to high density at the queue, occupants will wait in the queue to enter the node.

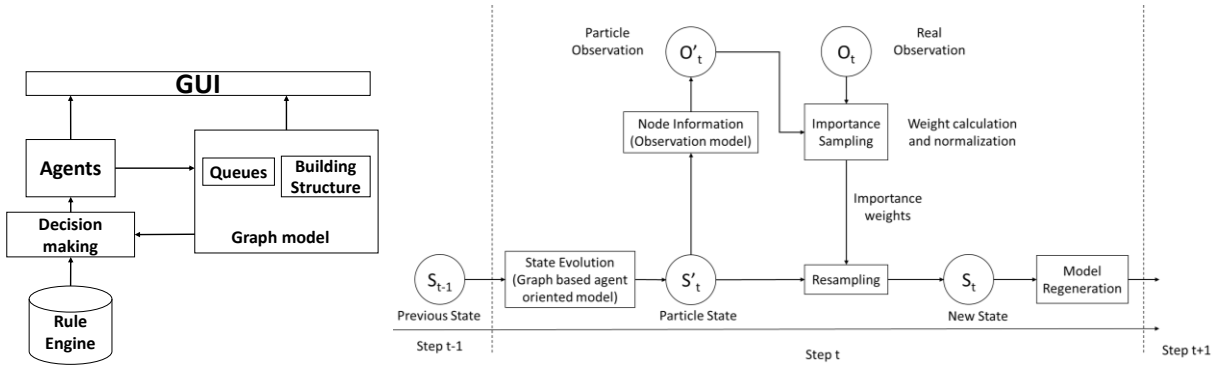

Figure 1: a) Graph-based agent-oriented framework b) PF-based data assimilation framework

To support data assimilation of the graph-based agent-oriented model, we used the basic bootstrap filter [14] as the algorithm for the data assimilation which is also known as SMC methods or particle filters. The observations are the occupancy information which is collected by various sensors installed in the rooms. In our framework, we consider only the node information as the state variable which reduces the size and complexity of the state for data assimilation. In the particle filters, particle represents a state, and each particle represents one of the possible states of the real system. During the sampling step, each particle goes

through the system transition function which evolves the system from the current system to a new state

$$S_{t+1} = G(S_t) + Q_t$$

In the above equation, G represents the state transition function based on graph-based agent-oriented model and Q_t is the system processing noise. The observation data for the particle filter is represented as $O_n = Obs(n) + m_t$. Here, $Obs(n)$ is the observation at room n and m_t is the measurement noise for the observation. Figure 1(b) represents the PF based data assimilation framework in which S_{t-1} is the previous state of the system which is evolved by the state transition function to new state S'_t which provides its observation (sensor data). This observation is compared with the real observation and weight computation and normalization are done in the importance sampling step. In the resampling step, particles are resampled based on the weights and the new state is estimated which is closer to the real system state. After that, there is a model regeneration step, which generates the required necessary properties of the system from the resampled state like agent and queue properties. The algorithmic structure of the standard particle filter with resampling is as follows:

Sequential importance resampling PF

Input: Occupancy states and corresponding importance weights at step t **{**S^i_t**,** w^i_t**}**

Output: Occupancy states and corresponding importance weights at step $t+1$ **{**S^i_{t+1} **,** w^i_{t+1}**}**

1. Sampling
   For each state S^i_t, draw a sample S^i_{t+1} from $p(S^i_t \mid S^i_{t+1})$ using $G(S^i_t)+Q_t$

2. Weight Updating
   For each state, update the weights $w^i_{t+1} = w^i_t \times p(O_{t+1}|S^i_{t+1})$

   Normalize $w^i_{t+1} = \frac{w^i_{t+1}}{\sum_{i=1}^{N} w^i_{t+1}}$

3. Resampling
   Draw N particles from S^i_{t+1} based on w^i_{t+1}

PARALLELIZATION OF SMC METHOD

Parallelization of an algorithm means to divide the tasks of an algorithm in such a way that they can be run in parallel and completed in faster time compared with running the tasks one after another. With the current abundance and low cost of multicore processor, it makes sense to utilize all the processing power of the hardware. Traditionally, designing a parallel program or converting from serial to a parallel program has been quite a difficult work due to complexities of managing thread synchronization, communication and sharing data. Now there are widely available parallel programming libraries, and they can be easily adapted to convert the serial programs to parallel. Previously, one of the main hindrance in converting programs to parallel would be if it would be worth the time to rewrite the complete serial program in a different language or different way to make it parallel. But currently various parallel libraries are available within almost all of the programming languages and it does not take much time and effort to convert to parallel.

Various levels of parallelism in a program can occur, of which task and data parallelism are two widely popular. Task parallelism refers to parallelizing the computer code across multiple processors (cores). It focuses on dividing the programming tasks to multiple processes or threads across available processors. Threads are the highest level of code executed by the processor. The number of threads depends on the cores of the central processing unit (CPU) machine. On the other hand, data parallelism refers to distributing the data in parallel so that they can be accessed by requiring programs in parallel and thus quickly.

In this work, we perform task parallelization, and we parallelize only the first step of the SMC algorithm, a sampling which is the most computationally costly step in the whole algorithm. According to Amdahl's law [15], in a program optimum speedup can be obtained only if we can successfully execute the task which consumes most resource in the algorithm. In the building occupancy simulation, there are complex states, and during sampling, each of the particles updates the state based on the simulation model. The remaining weight update and resampling steps do not involve complex state updates, and so they do not consume high resource. As such, in the SMC algorithm, sampling is the most resource-consuming step where hundreds to thousands of particles are executing independently. We can parallelize this sequential step and have each state of the particles execute separately in parallel to obtain speedup. Since each of the particles is independent, the states do not communicate with each other. This removes one of the complexities of a parallel program: managing communication between parallel threads. The next step, weight updating requires weights from each of the particles, so the main program will need to wait for each of the tasks to complete before moving to step 2. However, since sampling is the most computationally costly step, parallelizing only this step can provide us with some significant gain in the efficiency of the complete program. The parallel SMC algorithm is as follows:

Sequential importance resampling PF with parallel sampling

Input: Occupancy states and corresponding importance weights at step t **{**S^i_t**,** w^i_t**}**
Output: Occupancy states and corresponding importance weights at step $t+1$ **{**$S^i_{t+1,}$ w^i_{t+1}**}**

1. Sampling
   Run in parallel
   For each state S^i_t, draw a sample S^i_{t+1} from $p(S^i_t \mid S^i_{t+1})$ using $G(S^i_t)+Q_t$
   Wait for all tasks to complete

2. Weight Updating
   For each state, update the weights $w^i_{t+1} = w^i_t \times p(O_{t+1}|S^i_{t+1})$

   Normalize $w^i_{t+1} = \frac{w^i_{t+1}}{\sum_{i=1}^{N} w^i_{t+1}}$

3. Resampling
   Draw N particles from S^i_{t+1} based on w^i_{t+1}

Figure 2 (a) below shows the sampling step in which each of the particles executes sequentially starting from the first to the last and then goes to the second part of the algorithm. In figure 2(b), a parallel sampling step is shown where the particles are divided and given to available cores to process. This allows the algorithm to utilize each of the core rather than waiting idly. Available parallel programming libraries allow us to distribute the particles across the threads conveniently. We used parallel programming library for Java from [16] which provides a lightweight library known as PCDPlib. It provides lambda-based APIs for conveniently using java library. We used two of the parallel programming constructs for our experiments: asynchronous task parallelism (*async* and *finish*) and parallel loops (*forall*) [17].

Asynchronous task parallelism was used to divide the total number of particles into two halves where one half would run asynchronously in parallel with the second half. The *finish* construct would perform bulk task synchronization for both the halves after which the algorithm would move to step 2. Thus, *finish* would allow for all the particles to complete their state advancement and provide weights for normalization. We also used *forall* API for parallelizing the sequential operations in the loop. In loops where the number of iterations is known in advance, *forall* can create a separate task for each iteration and run them in parallel. The API handles the parallelization for each task if it is legal to do so and thus thread managements is not

required from user. Since *forall* allows for each particle to execute in parallel, using *forall* is efficient in case of parallezing sampling step of SMC algorithm.

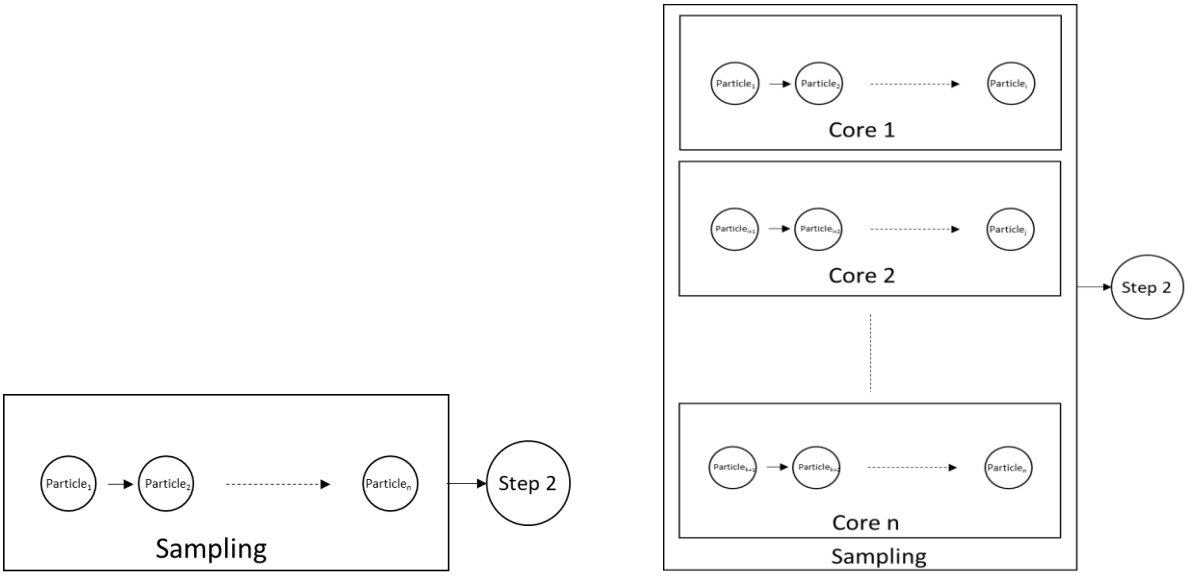

Figure 2: a) Sequential sampling b) Parallel sampling

EXPERIMENTS AND RESULTS

For the experiments, we ran the simulation in a building layout as shown in figure 3(a). The structure is a large building containing 80 nodes (rooms) where the capacity of each room is 100 occupants, and the width of the door is 25m. In the simulation, occupants enter the structure through the left terminal node 0, and the right terminal node 29 and most of the occupants go to node 113 and 195. Figure 3(b) shows a snapshot of the simulation for the simulation performing real time data assimilation using sensor nodes and SMC algorithm. The SMC algorithm gets data in real time from the sensors in the building, and it estimates the occupancy. In the figure, each dot represents an occupant agent, where the red color represents occupants moving through the nodes, blue color represents occupants reached at their destination and blue color represents the congestion created when occupants are trying to enter a room but the room is already full of capacity. The accuracy of the simulation model for real time estimation has been discussed in [xxxx], in this work we discuss only the runtime of the model for serial and parallel implementations. The simulation was run only to represent a total time of three real world hours in which occupants start arriving from the first hour and their rate gradually increase on the second hour.

Figure 3: a) Building structure b) Occupants being simulated in the building structure

We performed the simulation in two basic machines where one of them had i5 processor with 4 CPUs, and another had i7 with 8 CPUs. These are the general laptop machines which are used normally for such kind of simulations. We simulated 100, 200, 300, 500 and 1000 number of particles. Simulation was run for multiple times, and their average was noted for the runtime of each particle. Figure 4 shows the comparison of the serial and parallel runtimes for a various number of machines using *async* and *forall* APIs in the two machines.

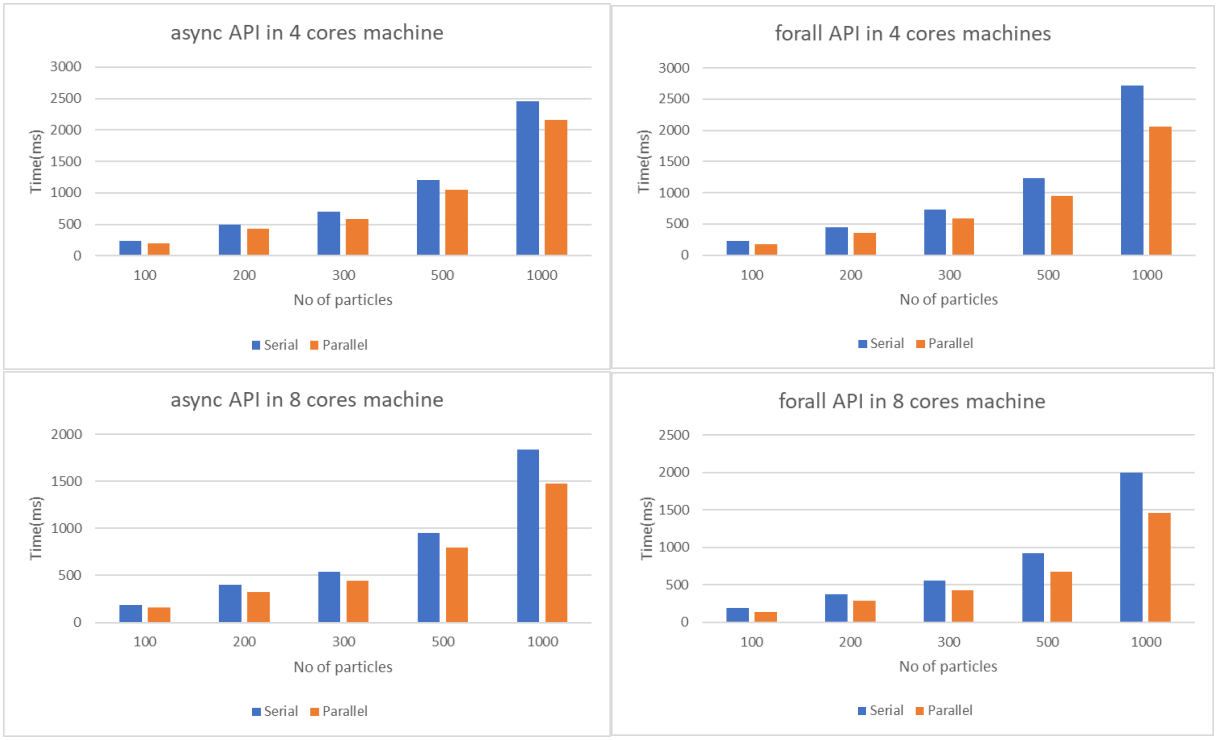

Figure 4: Comparing serial and parallel runtimes a) async API in a machine with 4 cores b) forall API in a machine with 4 cores c) async API in a machine with 8 cores d) forall API in a machine with 8 cores

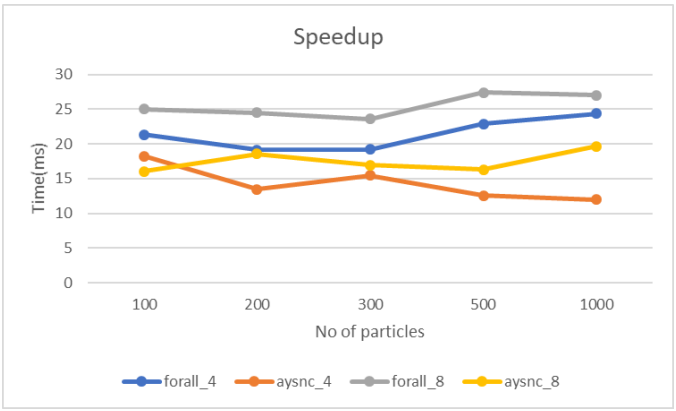

Figure 5: Speedup for the APIs in both machines

From the comparison in figure 4, we observe that in each case parallel runtime is better than the serial as expected. On average, a speedup of 14% and 21% was observed in the machine with 4 cores using *async* and *forall* APIs while in the machine of 8 cores, speedup of 17% and 25% was observed. We also observed that the speedup for *forall* API is better than *async* and the reason is that *async* executes only half of the

particles in parallel while the remaining half is executed in sequential. Whereas *forall* executes all the particles in parallel thus performing the maximum utilization of the hardware. Also, we observed that as the number of particles increases the increase in speed up also increases since more particles could be executed in parallel and faster. Figure 5 summarizes the speedup for both the APIs in two machines. It is evident that *forall* API has the highest speedup when using a machine with 8 cores.

DISCUSSIONS

The experiment results show that we can conveniently parallelize the SMC algorithm written in Java using the PCDPlib library APIs. We were able to achieve a speedup of up to 25% by using an 8-core computer which nowadays is easily accessible to everyone. In this work, we parallelized only the sampling step of the main SMC algorithm. In a real time building occupancy simulation model, there are mainly two components: first the building simulation model and second the data assimilation model which is making the real-time estimation. Since both components are dependent on each other the overall efficiency of a building simulation model depends on the runtime of both components. As such, a limitation of the model is that as the system becomes larger, the complexity of both the components increase and even if we decrease the run time of SMC algorithm, it might be insignificant compared to the run time of the overall simulation. Also, from the results we observe there is increase in efficiency as the number of particles increase, but after a certain threshold we might not be able to see any speedup since all the cores will be exhausted and not free to do work.

A solution for these issues would be to use multi-cluster and Graphical Processing Units (GPUs) which provide higher number of cores for the processing. Currently a sophomore student is assisting with this conversion. The main challenge will be conversion of the overall system to the one which can take advantage when running in large number of cores in multi-cluster or GPU. Since program in GPUs needs to be written in a different language (CUDA or similar other) than Java, there is a significant work to convert the current system. The student is currently in preliminary phase of research and working on understanding the building simulation model and parallel concepts. Student progress against the planned timeline will allow us to obtain a measure of the student learning in simulation modeling and parallel programming. We hope to complete these works by next year.

CONCLUSION

In this work, we parallelized the sampling step of the SMC algorithm and were able to achieve up to 25% speedup in the runtime when using a machine with 8 cores. Currently, the convenience of converting a serial program to parallel and availability of multicore machines makes it quite sensible that the first approach to make any algorithm efficient is to take advantage of parallel computing. In the future, we plan to run experiments in multi-cluster machines. We also plan to consider modifying the algorithm so that it can take advantage from high computing powers of GPUs.

REFERENCES

[1]. E.D. Kuligowski, R.D. Peacock, B.L. Hoskins, A review of building evacuation models, Gaithersburg, MD: US Department of Commerce, National Institute of Standards and Technology, 2005.

[2]. C. Wang, D. Yan, Y. Jiang, A novel approach for building occupancy simulation, Building simulation, Vol. 4, No. 2, Tsinghua Press, 2011.

[3]. Rai, S., & Hu, X. (2018). Building occupancy simulation and data assimilation using a graph-based agent-oriented model. *Physica A: Statistical Mechanics and its Applications*, *502*, 270-287.

[4]. Feng, X., Yan, D. and Hong, T. 2015. "Simulation of occupancy in buildings". Energy and Buildings, 87, pp.348-359.

[5]. Maasoumy, M. 2009. "Estimation of Occupancy Distribution in Buildings". Univ. Calif. Berkeley.

[6]. Cook, D. and Das, S.K., 2004. Smart environments: Technology, protocols, and applications (Vol. 43). John Wiley & Sons.
[7]. Xue, H., Gu, F. and Hu, X. 2012. “Data assimilation using sequential monte carlo methods in wildfire spread simulation”. ACM Transactions on Modeling and Computer Simulation (TOMACS), 22(4), p.23.
[8]. Gustafsson, F., Gunnarsson, F., Bergman, N., Forssell, U., Jansson, J., Karlsson, R. and Nordlund, P.J. 2002. “Particle filters for positioning, navigation, and tracking”. IEEE Transactions on signal processing, 50(2), pp.425-437.
[9]. Wang, M. and Hu, X. 2013. “Data assimilation in agent based simulation of smart environment”. In Proceedings of the 1st ACM SIGSIM Conference on Principles of Advanced Discrete Simulation (pp. 379-384). ACM.
[10]. Rai, S. and Hu, X. 2013. “Behavior pattern detection for data assimilation in the agent-based simulation of smart environments”. In Proceedings of the 2013 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT)-Volume 02 (pp. 171-178).
[11]. Rosenthal, Jeffrey S. "Parallel computing and Monte Carlo algorithms." *Far east journal of theoretical statistics* 4.2 (2000): 207-236.
[12]. Murray, Lawrence M., Anthony Lee, and Pierre E. Jacob. "Parallel resampling in the particle filter." *Journal of Computational and Graphical Statistics* 25.3 (2016): 789-805.
[13]. Li, Tiancheng, Shudong Sun, Miodrag Bolić, and Juan M. Corchado. "Algorithm design for parallel implementation of the SMC-PHD filter." *Signal Processing* 119 (2016): 115-127.
[14]. Doucet, A., De Freitas, N., & Gordon, N. (2001). An introduction to sequential Monte Carlo methods. In Sequential Monte Carlo methods in practice (pp. 3-14). Springer, New York, NY.
[15]. Amdahl, G. M. (1967, April). Validity of the single processor approach to achieving large scale computing capabilities. In Proceedings of the April 18-20, 1967, spring joint computer conference (pp. 483-485). ACM.
[16]. Imam, Shams and Grossman, Max, “PCDPlib Source Code,” https://github.com/habanero-rice/PCDP.
[17]. “PCDPlib Javadocs,” https://habanerorice.github.io/PCDP/.