J. Indian Inst. Sci.
A Multidisciplinary Reviews Journal
ISSN: 0970-4140 Coden-JIISAD

REVIEW ARTICLE

Architecture-Aware Modeling of Pedestrian Dynamics

Mehran Sadeghi Lahijani[1], Rahulkumar Gayatri[2], Tasvirul Islam[3], Ashok Srinivasan[3] and Sirish Namilae[4]*

Abstract | The spread of infectious diseases arises from complex interactions between disease dynamics and human behavior. Predicting the outcome of this complex system is difficult. Consequently, there has been a recent emphasis on comparing the relative risks of different policy options rather than precise predictions. Here, one performs a parameter sweep to generate a large number of possible scenarios for human behavior under different policy options and identifies the relative risks of different decisions regarding policy or design choices. In particular, this approach has been used to identify effective approaches to social distancing in crowded locations, with pedestrian dynamics used to simulate the movement of individuals. This incurs a large computational load, though. The traditional approach of optimizing the implementation of existing mathematical models on parallel systems leads to a moderate improvement in computational performance. In contrast, we show that when dealing with human behavior, we can create a model from scratch that takes computer architectural features into account, yielding much higher performance without requiring complicated parallelization efforts. Our solution is based on two key observations. (i) Models do not capture human behavior as precisely as models for scientific phenomena describe natural processes. Consequently, there is some leeway in designing a model to suit the computational architecture. (ii) The result of a parameter sweep, rather than a single simulation, is the semantically meaningful result. Our model leverages these features to perform efficiently on CPUs and GPUs. We obtain a speedup factor of around 60 using this new model on two Xeon Platinum 8280 CPUs and a factor 125 speedup on 4 NVIDIA Quadro RTX 5000 GPUs over a parallel implementation of the existing model. The careful design of a GPU implementation makes it fast enough for real-time decision-making. We illustrate it on an application to COVID-19.

Keywords: *Pedestrian dynamics, GPU, Architecture aware modeling*

1 Introduction

The spread of infectious diseases arises from complex interactions between disease dynamics and human behavior[6]. In recent years, pedestrian dynamics has found increasing application in simulating human movement patterns during the analysis of infection risk in crowded locations[5,14,22]. Pedestrian dynamics simulates the movement of individuals in a crowd, from which social proximity can be characterized. An

[1] *Department of Computer Science, Florida State University, Tallahassee, USA.*
[2] *Lawrence Berkeley National Lab, Berkeley, USA.*
[3] *Department of Computer Science, University of West Florida, Pensacola, USA.*
[4] *Aerospace Engineering Department, Embry-Riddle Aeronautical University, Daytona Beach, USA.*
**asrinivasan@uwf.edu*

infection spread model then uses the social proximity information to estimate infection risk for directly transmitted diseases.

Social-force models for pedestrian dynamics are the most popular ones for simulating trajectories of individual pedestrians. Their computational structure is similar to that of an N-body problem, involving the integration of Newton's laws of motion: $md^2r(t)/dt^2 = Force(t)$, where r is the position of a person with mass m at time t. The force term is not modeled as the actual physical force. Rather, models for human behavior are designed so that one term generates a "force" that propels a person toward that person's destination, while a counteracting force term limits the speed due to the presence of others nearby or because of fixed surfaces through which a person cannot move, such as walls. Solving the above differential equation generates a trajectory for each pedestrian.

The trajectories can then be analyzed further to determine quantities of interest to the application. For example, the Self-Propelled Entity Dynamics (SPED) model for pedestrian dynamics has recently been used to analyze infection spread in airplanes[5,14,15,22]. It counts the number of contacts between people from the trajectories and uses it to estimate the risk of infection spread. This work was identified as one of the twelve major scientific breakthroughs using the flagship Blue Waters supercomputer at the National Center for Supercomputing Applications (NCSA)[12].

One of the challenges in modeling arises from inherent uncertainty in human behavior, which makes predicting any specific outcome difficult. A single simulation does not capture the diversity of outcomes arising from variations in human behavior. A recommended approach to dealing with this problem is to parameterize the sources of uncertainty—such as the natural speed of a pedestrian in the absence of others nearby—and carry out a parameter sweep to generate a large number of possible scenarios[5,15,22]. The number of scenarios is large in situations that deal with extreme events, such as stampedes or epidemics, where we wish to capture a wide range of outcomes so that it would include the extreme events that one desires to observe. This leads to a high computational cost.

The computational effort is currently handled through efficient parallelization of these models on massively parallel machines and GPUs[5,10,19,22,24]. However, the times obtained are inadequate for real-time decision-making. For example, Chunduri et al.[5] observe that results are often needed in the order of a minute during a decision meeting to avoid discussions digressing in other directions. Some large SPED simulations take the order of ten minutes on massively parallel processes using the order of 30,000 cores, *without taking queue wait time into account*[22]. This limitation was handled by precomputing the results and performing only the analyses in real-time in our more recent work[5]. However, this approach cannot help in analyzing pedestrian movement for new procedures or policies that have not been precomputed. Rather, it can only help answer queries related to new epidemics but with the precomputed trajectories.

We propose a new approach that can enable social force-based pedestrian dynamics models to meet the real-time simulation constraints required for decision-making in an emergency. It is based on the following key observations on the limitations of the previous approaches. (i) Current optimizations develop efficient implementations for existing models. Such an approach is required in physical sciences, where the models precisely capture the behavior of natural processes. Human behavior cannot be captured with that level of accuracy, and consequently, we have more leeway to modify the model to suit the computational architecture. (ii) Current optimization approaches attempt to reproduce the results of each simulation from the unoptimized code. However, the results of a single simulation are not meaningful in the application context. Rather, the output of the entire parameter sweep is the meaningful result. We develop a model that will lead to good computational performance while producing results that are accurate over the entire parameter sweep.

We discuss our new Constrained Linear Movement (CALM) model. Its name indicates our target of simulating the movement of pedestrians in narrow passageways, with this paper applying it to passenger movement when deplaning from an airplane. However, the social force model can be used in a wider context, although our software implementation targets movement in narrow passageways. It has been designed to perform efficiently on GPUs but also yields substantial performance improvement on CPUs. It has been designed to yield good computational performance not just on a single simulation but also on the parameter sweep.

We design this model by considering the application needs and the computational requirements simultaneously. This contrasts with the traditional approach, where the model is developed first based on application needs and then optimized on different architectures. Design considerations for performance include those

for the computational effort and data size, which impacts the amount of parallelism in the parameter sweep due to memory limitations of GPUs. We obtain a factor of around 125 improvement in performance over SPED when we use 4 GPUs on a single GPU node of the Frontera supercomputer, which is fast enough to meet the real-time requirements for use in decision-support systems during emergencies. Furthermore, we design a hybrid CPU–GPU version that uses number-theoretic properties of the parameter sweep for efficient load balancing and increases the performance of the parameter sweeps.

Our primary contributions lie in proposing and demonstrating the effectiveness of architecture-aware modeling for pedestrian dynamics, optimized for both individual simulations and for a parameter sweep. This approach can have a transformative impact on enabling effective use of GPUs for other models dealing with human behavior because these models typically do not capture human behavior so precisely that one needs to replicate the results of the original model exactly. This is particularly critical in dealing with emergencies, such as the current COVID-19 pandemic, because the human response to the emergency and public policy choices need to be understood to develop effective interventions to mitigate the crisis.

The outline for the rest of the paper is as follows. We introduce pedestrian dynamics and the SPED model in "Pedestrian Dynamics and the SPED Model". The CALM model is introduced in "CALM Model", along with the architectural characteristics of GPUs relevant to its design. This is the most significant section in this paper, explaining our novel contribution. We provide empirical results on the performance of the CALM model on CPUs and GPUs in "Performance Evaluation". We then discuss the validation of the model and its application to analyzing COVID-19 spread in "Validation" and "Application to COVID-19", respectively. We finally present related work in "Related Work" and summarize our conclusions in "Conclusions"

2 Pedestrian Dynamics and the SPED Model

We first introduce pedestrian dynamics and then provide details on the SPED model. We then identify its computational limitations.

2.1 Pedestrian Dynamics

Pedestrian dynamics models have been developed based on various approaches such as fluid flow[11], cellular automata[3], queuing theory[17], molecular dynamics-based social force models[8], and machine learning from videos[1].

Pedestrian dynamics-based on social force models have gained the most popularity recently and are best suited for individual trajectory evolution. These often adapt methods from molecular dynamics[2], which simulates material behavior at atomistic scales, by developing social force as analogs to inter-atomic forces[8]. The SPED model is one such model and has been used to study infection spread in air travel[5,14,15,22].

This approach models both mobile pedestrians and stationary objects, such as walls and seats, as particles, with multiple particles needed for large objects such as walls. The trajectory of a pedestrian evolves due to the pedestrian's progress toward a goal and due to interactions with other pedestrians and stationary objects. The force f_i acting on the ith pedestrian with mass m_i at any time t is defined by the following equation:

$$f_i = m_i \frac{(dv_i)}{dt} = m_i \frac{(d^2 r_i)}{(dt^2)} = \frac{m_i}{t_c}(v_0 i - v_i) + \Sigma_{(j \neq i)} f_{ij}, \quad (1)$$

where r_i is the current pedestrian position, v_{0i} is the desired velocity of pedestrian i in the absence of anyone else, v_i is the actual velocity, and t_c is a time constant. The first equality arises from Newton's laws of motion, while the last equality is a model's description of human behavior. The first term in the human behavior is a self-propulsion force for momentum generated by a pedestrian's intention, while the second represents a counteracting repulsive force that is the sum of the forces due to other pedestrians, with pedestrian j contributing a force f_{ij} on pedestrian i. Social-force models typically differ in their definition of f_{ij} [8, 14] which is normally driven by differing application contexts.

2.2 SPED Model

The SPED model was developed to simulate the movement of passengers in an airplane. It is based on a molecular dynamics code by Brenner et al.[2]. The self-propulsion term is identical to that in Eq. (1).

The repulsive forces f_{ij} are determined as follows. Let the distance to the nearest passenger or obstacle in i's direction of motion be d_i. If d_i is greater than a certain threshold, then $f_{ij} \leftarrow 0$ for all j. If it is less than another threshold, then f_{ij} is the negative of the gradient of the Lennard–Jones potential defined by Eq. (2), where r_{ij} is the

distance between i and j, and ϵ and σ are some constant parameters.

$$V_{ij}^{LJ} = \varepsilon \frac{\sigma}{r_{ij}^{12}} \tag{2}$$

If d_i is between the above two thresholds, then a gradual reduction in speed is applied: $v_i(t+\Delta t) \leftarrow \alpha_i\, v(t)$, where α_i is defined by Eq. (3), where λ is a constant representing the desired stopping threshold of the passengers:

$$\alpha_i = 1 - \frac{\lambda}{d_i} \tag{3}$$

Certain human behavioral features are added into the SPED model through code outside of Newton's laws. For example, passengers take some time to stow their luggage during boarding or retrieving their luggage during deplaning. Passengers also typically allow others in rows ahead of them to get into the aisle before moving forward.

Algorithm 1 provides a high-level description of SPED. The following details of the algorithm are relevant to its performance. (i) Lennard–Jones is a short-range force that is negligible beyond a certain threshold. A neighbor list for each passenger keeps track of all other passengers within a threshold. Rather than sum this force over all passengers in the plane, the code sums it only over the passengers close by. The neighbor list needs to be updated every iteration, even if the Lennard–Jones force is not used in that iteration. That is the reason for Lennard–Jones being called each iteration. (ii) This model uses a third-order Nordsiek solver, for which three derivatives of the position (velocity, acceleration, and rate of change of acceleration) are required. (iii) Position update is performed only for passengers who are in a position to move based on their behavioral characteristics, as mentioned above. The model has been validated by comparisons with empirical data and has been successfully applied to airplane boarding and deplaning[14] pedestrian queues in theme parks and airport security checks[9].

2.3 Performance Optimizations and Their Limitations

We now summarize performance optimizations that have been performed on SPED and the limitations of this optimized code, making it inadequate for decision-support meetings during emergencies. These optimizations arise from (i) optimizing an individual simulation and (ii) performing the parameter sweep more efficiently.

The original SPED code took a few hours per simulation. Conventional sequential code optimization, followed by an application-specific workflow optimization, led to an order of magnitude improvement in performance[22], with maximum computation time being around 20 min for a single simulation on the Blue Waters machine at NCSA.

The parameter sweep was parallelized by having each simulation run on one core, although a single core could run several simulations. Simulation times vary widely depending on the choice of parameter values. Dynamic load balancing was used to deal with this, and parallel I/O optimization yielded additional efficiency, leading to parallel efficiency over 90%[22].

In subsequent work[5], the parameter sweep itself was improved. The original parameter sweep used a lattice of points. For example, if there are 10 choices for parameter 1 and 10 for parameter 2, then all 100 combinations of parameter values would be used. A low discrepancy sequence[13] can cover the parameter space more efficiently. It could reduce the number of parameter combinations required by one to three orders of magnitude in simulations with five parameters.

Algorithm 1: SPED algorithm

```
while (There are passengers remaining in the plane) do
    for (each passenger in the plane) do
        Compute self-propulsion force
        Compute neighbor list
        Compute Lennard-Jones potential
        Find the nearest neighbor in direction of motion
        if (Nearest neighbor is too close) then
            Use repulsive force computed by the Lennard-Jones potential
        else if Nearest neighbor is too far then
            Repulsive force = 0
        Else
            Gradually decrease the speed of the passenger based on the distance to the nearest
            entity
        end if
    end for
    for (each passenger allowed to move) do
        Update the velocity and position using Nordsiek solver
    end for
end while
```

The computation time is still limited by the time for the slowest simulation to around 20 min on Blue Waters and 5 min on Frontera. Furthermore, to achieve the above limit on a parameter sweep, one would need hundreds of cores. The above work handled this issue by precomputing the passenger trajectories and performed only the analysis, such as determining the number of contacts in real-time. This is inadequate for decision meetings where new policies, which could impact passenger movement patterns, need to be examined.

Furthermore, adequately accurate results require thousands of simulations. This would require several nodes of a supercomputer and even more if individual simulations were parallelized. The queue wait time on a supercomputer itself would make it infeasible for real-time results. Our goal is to design a model where such a parameter sweep could be performed on a dedicated in-house system accelerated with GPUs.

3 CALM Model

This section explains our novel contributions. We first describe GPU architectural features of relevance to our model design, then define the CALM model, and finally discuss its GPU implementations.

3.1 GPU Architectural Features

We use NVIDIA GPUs and CUDA for programming and, therefore, adopt their terminology. GPUs have several Streaming Multiprocessors (SMs), each capable of running thousands of threads in parallel. For example, the NVIDIA Quadro RTX 5000 GPUs have 48 SMs.

A group of threads forms a block, and each block is assigned to one SM for execution, although an SM might handle several blocks. Each GPU has a global memory (DRAM) that is accessible to threads running on all SMs. Each SM has a shared memory that is allocated per block, with this memory being low-latency. Only threads running on a block can access the shared memory allocated to that block. The NVIDIA Quadro RTX 5000 GPU has 16 GB of global memory (DRAM) and up to 64 KB of shared memory per SM, which is shared with the L1 cache. Among other factors, the amount of shared memory required by each block limits the number of blocks that can simultaneously run on each SM.

3.2 Model Design Features

Any pedestrian dynamics model clearly needs to capture human movement features correctly. We have discussed application-related aspects, such as validating that the model correctly simulates human movement, in a separate paper[21]. In the current manuscript, we identify the performance-related features used to create the model on the GPUs. Certain crucial aspects of the model design are to (i) enable massive parallelism, (ii) reduce thread divergence, and (iii) reduce the data movement overhead.

GPUs rely on deploying a large number of threads to hide data access latency. Limitations in the shared memory size impact the number of blocks that can run simultaneously, and consequently the parallelism. It is, therefore, important to reduce the amount of memory needed.

SPED uses the third-order Nordsiek solver[2] for the numerical solution of the ordinary differential equation, which is required for the high accuracy and energy conservation desired in molecular dynamics simulations. A disadvantage of the Nordsiek solver is that it requires three derivatives of position, even though Newton's law does not require the third derivative. This increases the memory required, which can be a bottleneck on the GPU, as explained later.

The mass of each passenger consumes additional memory, which is potentially not useful. For example, it is fairly common in pedestrian dynamics to use the same mass for each pedestrian. A formulation that avoids the use of the mass would reduce the memory required.

Thread divergence arises when different threads in the same warp (a subset of the threads in a block) take different execution paths. In the pedestrian dynamics computation, it is natural to assign a passenger to one thread. This would lead to thread divergence in the SPED model because the computation of the repulsive forces can take substantially different paths for each passenger, depending on the distance to the nearest passenger on the path (d_i). Furthermore, repulsive forces are the computational bottleneck, with the Lennard–Jones potential itself consuming around 80% of the time in the SPED model. It is desirable for the model to use a single equation irrespective of the value of d_i.

3.3 CALM Model

In the CALM model, rather than solving Newton's law of motion directly, we reformulate it to describe the evolution of the acceleration so that

the mass does not need to be stored. It is also intuitively appealing because the acceleration of passengers in a plane is likely a behavioral characteristic rather than being governed by their mass. Furthermore, we perform a parameter sweep on the model parameters, which would cover different behavioral tendencies. The model is given by the following equation:

$$\frac{(d^2x_i)}{(dt^2)} = \frac{(\beta_i v_0 i - v_i)}{t_c}. \tag{4}$$

The value of β_i is a function of d_i. It satisfies the following properties. (i) If d_i is large, it will tend to make the pedestrian move toward that pedestrian's desired speed. (ii) If d_i is very small, it will drastically decrease the pedestrian's velocity. (iii) If d_i is between the two extremes, it will work as a slight decelerator. An expression for β_i of the form given in Eq. (5) satisfies these properties, with constants $a=2.11$, $b=0.366$, and $c=0.966$ selected to capture realistic human behavior. This is explained in[21], because it deals with the behavioral rather than performance aspect of the model.

$$\beta_i = \mathrm{c} - \mathrm{e}^{-\mathrm{a}(\mathrm{d_i}-\mathrm{b})} \tag{5}$$

Instead of the Nordsiek scheme for the solution of the CALM model, we use the Euler method. This has been used successfully by other research groups in pedestrian dynamics[7]. (This detail is available on the following website by its author that provides further details on that paper: http://angel.elte.hu/panic). It avoids storing an extra derivative, thus reducing the memory required.

The CALM algorithm is presented in Algorithm 2. It includes all behavioral features of the SPED model in its code (which is not shown in the algorithm below). In addition, it avoids deadlock that could arise in the SPED model. This happens when two persons try to reach a location that cannot accommodate both of them, such as the center of the aisle in a plane. In that case, one person wins the race, with a random component to this decision. This step is explicitly coded.

Comparing Algorithm 2 and Algorithm 1 reveals the crucial differences between the CALM and SPED algorithms. First, using Eq. (4) and the Euler solver instead of Lennard–Jones potential and third-order Nordsiek solver, we decrease the computational overhead significantly. Second, we encode different repulsive forces in one formula Eq. (4), which reduces the thread divergence on the GPUs. Third, instead of keeping a neighbor list for each passenger, we find the nearest entity (passenger or physical-obstacle), which accurately captures pedestrian movement in airplanes[21]. These optimizations also result in less data movement and thus better performance on the GPUs.

3.4 CALM Parameters

We define six parameters to model uncertainties in the human behavior. The first parameter is the average walking speed of passengers which is in the range of 1.1–1.3 m/s[25], and specifies the maximum reachable speed for each passenger. Passengers walk with this speed when there are no obstacles in their path. However, we identified three situations in which passengers do not walk with this speed and, therefore, define coefficients to multiply in this average speed. When passengers move toward the overhead bins, their maximum reachable speed is multiplied by a coefficient in the range of 0.2 to 0.6. When passengers want to align themselves on the center of the aisle, their maximum reachable speed is multiplied by a coefficient in the range of 0.2 to 0.7. Passengers in the aisle wait for the passengers in front of them to move first. A passenger can move in the aisle if their distance to the nearest passenger in front of them is more than a parameter in the range of 0.5 to 1.6 m. When passengers get closer than a threshold (0.2–1.5 m) to the end of the intersection of the main aisle and the exit aisle, their maximum speed threshold is multiplied by a parameter in the range of 0.2 to 0.8 while they are turning toward the exit door.

3.5 CALM Implementation on GPU

3.5.1 Data on DRAM

The CALM model has several feature arrays. We first consider the case when these are present in the GPUs DRAM.

Algorithm 2: CALM algorithm

```
while (There are passengers remaining in the plane) do
    for (each passenger i in the plane) do
        Find the nearest neighbor in direction of motion and compute d_i
    end for
    for (each passenger i in the plane) do
        Compute β_i
        Compute the right hand side of equation (4)
        Update the velocity and position using an Euler solver
    end for
end while
```

Each pedestrian is assigned to one thread, although, in principle, one thread could handle M pedestrians, where M is the ratio of the number of pedestrians to the number of threads. We will discuss the process of tuning M for our application in the experimental setup section, "Performance Evaluation", of this article.

As Algorithm 2 shows, computing for a passenger at each time step requires the most recent position of other passengers (to find the correct nearest passenger, for example). Therefore, synchronization of all threads after each iteration is necessary. To avoid inter-block synchronization, we use a single block for each simulation. As a consequence, synchronizations will happen only among threads on the same block, which makes this process feasible without solutions that would incur significant overhead. On the other hand, it decreases the efficient use of the GPU for a single simulation. However, our goal is to use the GPU efficiently for the parameter sweep. While a single simulation would keep only one SM occupied, the parameter sweep keeps the whole GPU occupied by concurrently running several simulations, one simulation per thread block. We use one kernel call per parameter sweep, with each using one block per simulation. Therefore, several simulations execute simultaneously on the GPU and make effective use of the whole GPU.

3.5.2 Shared Memory Implementation

We also develop a shared memory version of the CALM model. Here, we copy the data into the shared memory before initiating the execution of the simulation. Therefore, we replace the accesses to the DRAM in the previous implementation with accesses to the shared memory during the course of the simulation. This implementation has potential advantages and disadvantages over the DRAM version. On the one hand, access to shared memory is much faster than access to the DRAM. On the other hand, the shared memory size is small, thus limiting the number of blocks that can run simultaneously. "Comparing CALM and SPED" evaluates the net impact of these two factors by comparing the shared memory and DRAM versions.

3.5.2.1 I/O The CPU version of the CALM model writes the positions of passengers to the output file every K iterations, where K is a suitably chosen constant. We used gprof to profile a single simulation on the CPU, and the profiling information showed that I/O on the CPU takes less than 0.01% of the execution time. I/O, thus, does not impose a noticeable cost in CPU simulations. However, a similar procedure on the GPU would incur a significant I/O overhead since it would involve several I/O calls with data movement over the PCI-express link. Instead, we write the positions of the passengers to an array every K iterations on the device. After the execution of the parameter sweep is finished and the kernel returns to the host, the host writes the whole array to the output files corresponding to each simulation. Therefore, data transfer from the device to host memory and subsequent I/O by the host has to happen only after the parameter sweep execution is finished and involves a few large I/O calls instead of many small ones.

3.5.3 Floating Point Precision

We have a choice of using different precision levels. While double-precision would be more accurate, our basic assumption is that no model captures human behavior so accurately that the result of a single accurate simulation is meaningful. Consequently, we observed that lower precision can lead to sufficiently accurate results for the parameter sweep. Half-precision is even faster than single precision, but its results were not correct. In fact, there was no progress in the simulations using half-precision because the numerical solution of the differential equation solver requires increments to the current values of variables that are very small and get rounded to zero.

We quantify the performance gain from the use of single-precision by implementing both single-precision and double-precision versions. There is potential for performance gains in single-precision due to better compute performance on it, especially on the GPU. Furthermore, on the shared memory implementation on GPUs, it increases the parallelism during the parameter sweep by decreasing the amount of memory required per block, which increases the number of blocks that can be assigned to each SM. While single-precision also increases I/O speed, the I/O overhead is not sufficiently high to contribute much to the performance improvement.

4 Performance Evaluation

4.1 Experimental Platform

We use the Frontera supercomputer at the Texas Advanced Computing Center for all our experiments. This system is equipped with 8008 Cascade Lake (Intel Xeon Platinum 8280) nodes and 90 GPU nodes and is ranked the 9th fastest supercomputer in the November 2020 Top 500

list. Each Cascade Lake node contains 56 cores and 192 GB of DDR4 RAM. Each GPU node has 4 NVIDIA Quadro RTX 5000 GPUs and 2 Intel Xeon E5-2620 v4 (Broadwell) CPUs with 8 cores (16 simultaneous threads) and 128 GB of DDR4 RAM per CPU.

The operating system on the Frontera supercomputer is CentOS Linux 7.6.1810, and we use g++ (GCC) 8.3.0, mpicxx (ICC) 19.0.5.281, mpifort (IFORT) 19.0.5.281 compilers and NVIDIA CUDA 10.1 compilers for compiling our codes.

We use the gettimeofday() function with 1 μs resolution for measuring the timings that we report in this section. We repeat all of our runtime measurements five times and report the minimum measured runtime to decrease the noise from the operating system jitter[4].

4.2 Code Availability

We have provided a public repository on Gitlab that contains all the codes and guidelines for reproducing our results. This repository can be accessed at https://gitlab.com/Mehran_SL/gpu-calm.

4.3 Simulation Details

All of our experiments simulate passengers' disembarkation from a full Airbus A320 with 144 seats. We employ a low-discrepancy parameter sweep using a scrambled Halton low discrepancy sequence[5]. We have also implemented a hybrid CPU–GPU version of the model to make use of the entire hardware resource. For the CPU simulations of this hybrid implementation, we use a master-worker algorithm to balance the load dynamically among the cores. CPU simulations start with one simulation per core, excluding the core where the master runs. The master assigns a new simulation to a core that has completed its previous simulation. The hybrid simulations, where CPU and GPU both run simulations, are more complicated due to load balancing issues that depend on number theory and are explained separately.

We tuned the number of threads per block in the GPU code by examining the performance of the CALM model on GPU with different numbers of threads. We observed that 144 threads—one thread per passenger—yield peak performance. Therefore, we use 144 threads for each simulation in our experiments, which gives us the maximum possible parallelism with having each thread doing the computations of one pedestrian.

4.4 Convergence Analysis

As we mentioned earlier, we perform a parameter sweep to capture several possible scenarios, including extreme, uncommon ones[21]. We need to determine a suitable size for the parameter sweep so that we sample the space well. On the other hand, taking too many samples would unnecessarily increase the computational load. We wish to find the minimum sample size N that ensures that the results are accurate and reliable. We use a convergence check to determine this using the methodology of[5]. Here, we consider histograms of quantities of interest and check whether moments of their distribution change less than a given threshold (5%) while the sample size doubles. Specifically, we use this criterion for the following two quantities. (i) Deplaning time, which shows the time in the real world for the deplaning process that is simulated. (ii) The total number of contacts between passengers during deplaning, which counts the number of people that come within a threshold distance of each other. We choose this to be 1.83 m (6 feet), which is the threshold corresponding to SARS-COV-2 transmission. We define N to be the number of simulations in the parameter sweep and examine the results for $N=2000$, $N=4000$, $N=8000$ and $N=16{,}000$ as shown in Figs. 1 and 2.

Convergence analysis of the moments, shown in Fig. 3, shows that the mean, standard deviation, skewness, and kurtosis have converged for $N=16{,}000$. Comparing the histograms in Figs. 1 and 2, we see that the results with $N=2000$ are qualitatively similar to those with $N=16{,}000$, while the results with $N=8000$ are also quantitatively close to $N=16{,}000$. Consequently, we use $N=2000$ for real-time results—when qualitatively accurate results are needed quickly during a decision meeting—and $N=8000$ when quantitatively accurate results are desired. The convergence check of the SPED model also shows that it produces quantitatively accurate results in the order of ten thousand simulations[5]. We emphasize that this analysis provides an insight into the convergence of the parameter sweep results, while the accuracy and validity of the model and simulations are discussed elsewhere[21].

4.5 Selection of the Best CALM Version

We wish to choose the best CALM version on the GPU to use for further study. We use $N=2000$ for selecting the best implementation and then evaluate that implementation further with $N=8000$. This ensures that evaluation is performed in a

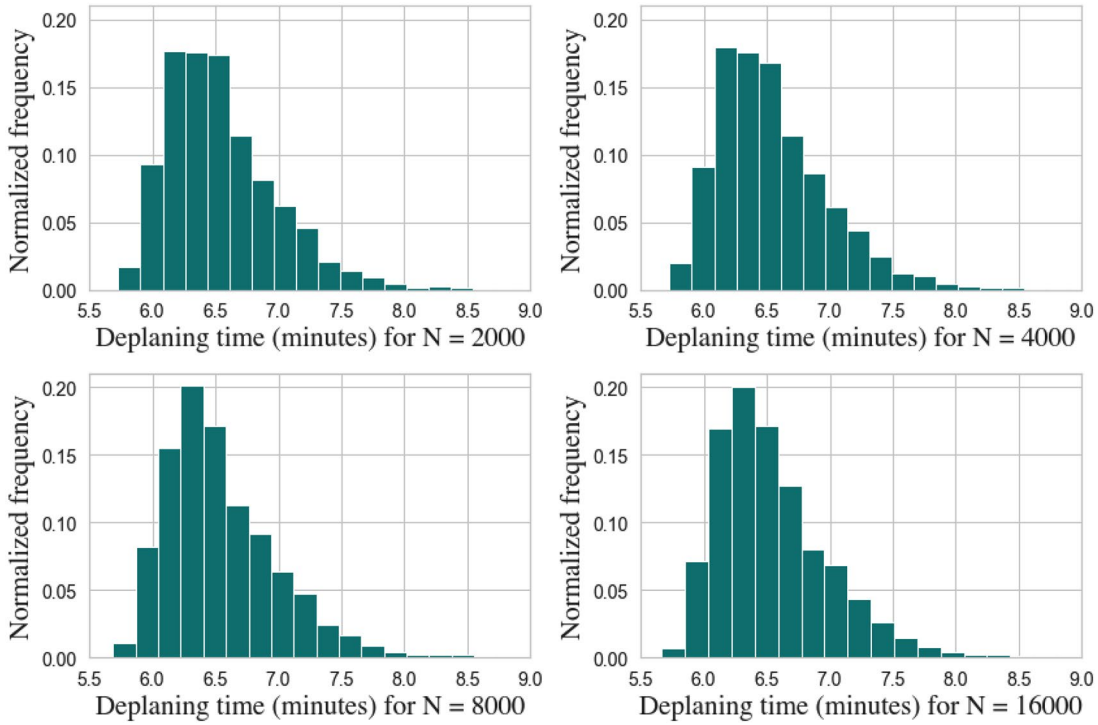


Figure 1: Histograms of deplaning times.

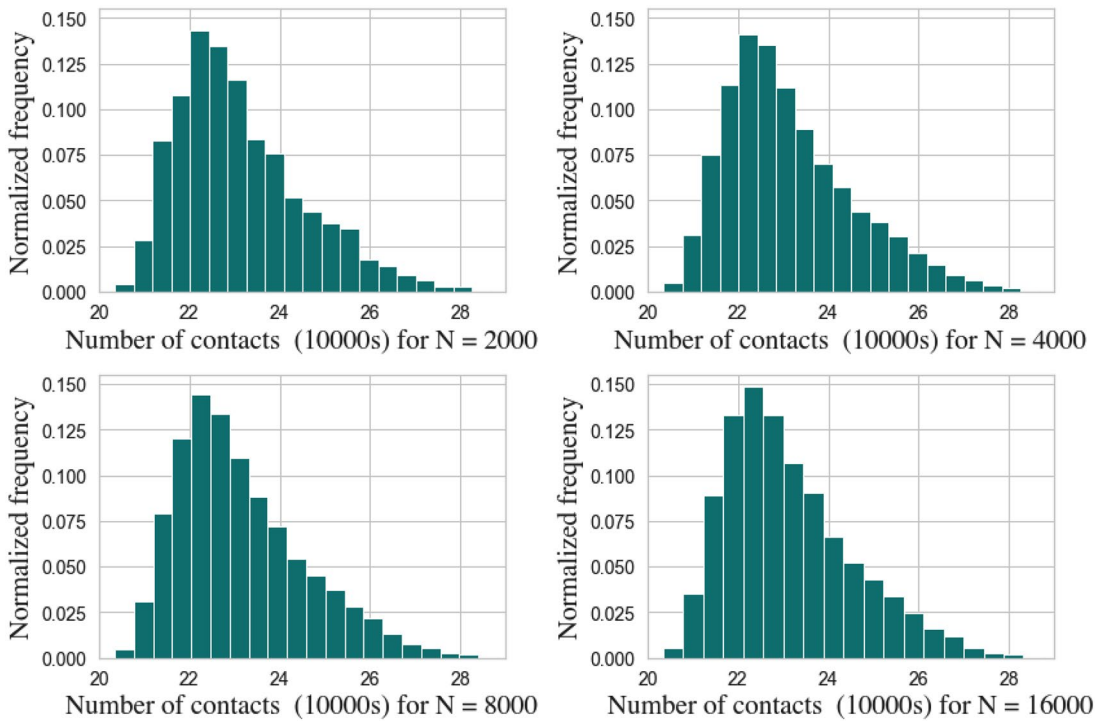
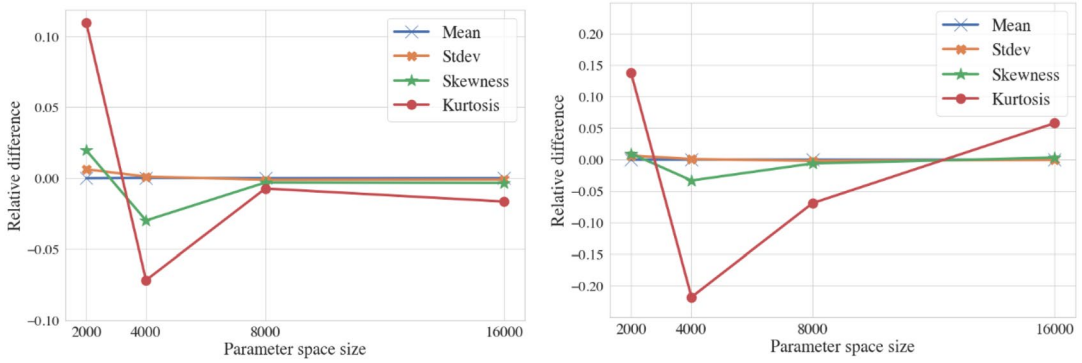


Figure 2: Histograms of the number of contacts.

different situation than the selection. We carry out the GPU experiments on one NVIDIA Quadro RTX 5000 GPU. We present the results of these four parameter sweeps in Table 1. These results show that the parameter sweep with data in DRAM is faster than shared memory.



(a) Statistical convergence check for deplaning time

(b) Statistical convergence check for the number of contacts

Figure 3: Statistical convergence check for the parameter sweeps. **a** Statistical convergence check for deplaning time; **b** Statistical convergence check for the number of contacts.

Table 1: Run-Time of parameter sweeps of the CALM model on one GPU with ***N*** = 2000.

Memory type	Floating-point precision	Run-Time (s)	STDEV
DRAM	Single-precision	257.09	1.18
DRAM	Double-precision	1189.10	14.27
Shared memory	Single-precision	369.01	0.99
Shared memory	Double-precision	1592.98	0.75

To better comprehend the performance of these GPU implementations, we also profile our code on the same system with the NVIDIA Nsight suite. We profile on parameter sweeps of size 500, and we see that again the parameter sweep on DRAM is faster. We observe a 29.24% decrease in achieved occupancy—the ratio of concurrent active warps—when we use shared memory instead of DRAM. While better occupancy does not necessarily mean better performance in all applications, in our application, we can see that each simulation (each thread block) uses slightly less than a third of the available shared memory (18.26 KB out of 65.54 KB). This is the main factor in decreasing occupancy. Our profiling results also reveal that the shared memory parameter sweep has around 26% less ALU utilization and 17% less Fused Multiply Add/Accumulate (FMA). These results also support our hypothesis about the reason for less occupancy in shared memory parameter sweeps. We conclude that fewer simulations can concurrently run when we use shared memory, which makes parameter sweeps on shared memory slower on these NVIDIA Quadro RTX 5000 GPUs.

We next assess the difference in theoretical occupancy in the DRAM and shared memory implementations with $N = 2000$. The average occupancy can be obtained as the ratio of the sequential time to parallel time. The latter is shown in Table 1. We used the clock() function to determine the runtime of each block in the kernel and added them to compute the total sequential runtime. With single-precision floating-point numbers, the average occupancy on DRAM is $\frac{48860}{257.09} \approx 190$, while it is $\frac{53380}{369.01} \approx 146$ on shared memory. Thus, DRAM increases the occupancy by approximately 30% in the single-precision implementation.

According to the theoretical performance of NVIDIA QUADRO RTX 5000 GPUs, peak FP32 (single-precision floats) performance is around 11.2 TFLOPS while the peak performance of FP64 (double-precision floats) is only about 348 GFLOPS[16,23]. Moreover, our profiling results show a 77% decrease in L1 cache throughput, a 70% reduction in L2 cache throughput, and a 91% decrease in DRAM throughput, and a 5% drop in SM throughput when we switch to double-precision numbers on shared memory.

Table 2: Comparison of SPED and CALM on parallel parameter sweeps with $N=8000$.

Model	Run-Time (s)	STDEV
SPED	31,947.8	189.56
CALM	975.4	8.10

Table 3: Performance of the CALM model when using multiple GPUs for parameter sweeps with $N=8000$.

Memory type	Number of GPUs	Run-Time (s)	STDEV
DRAM	2	497.25	4.61
Shared memory	2	563.30	4.01
DRAM	4	256.36	7.36
Shared memory	4	307.12	4.09

Moreover, a 70% increase in the amount of shared memory needed by each block leads to a 75% decrease in occupancy for these parameter sweeps. We infer that the lower computation and memory throughput are the major factors in the performance drop of parameter sweeps that use double-precision floating-point numbers. We conclude that the single-precision DRAM is the best GPU implementation for the system we are using, while the shared memory implementation could perhaps perform faster on some other architectures. All subsequent tests in this article use single-precision and DRAM on the GPU.

4.6 Comparing CALM and SPED

We have previously shown that a CPU implementation of the CALM model outperforms the SPED model by a factor of 58.7 on 56 CPU cores (one node) with two Xeon Platinum 8280 28C processors of the Frontera supercomputer when we compare parameter sweeps of size 1000[21]. In this article, we want to show the effectiveness of architecture-aware modeling. Therefore, we compare each model's performance on the architecture that each model was, respectively, designed for (multicore-CPU for SPED and GPU for CALM). In our next experiment, we run parameter sweeps with $N=8000$ to compare the performance of the SPED model (which has only a CPU implementation) against the GPU implementations of the CALM model. We carried out a parallel low-discrepancy parameter sweep of the SPED model on 56 cores of a Cascade Lake (Intel Xeon Platinum 8280) node on the Frontera supercomputer. The results of this experiment are shown in Table 2. The key observation in these results is that the CALM model significantly outperforms the SPED model by a factor of 32.

4.7 Using Multiple GPUs

Modern architectures enable the use of multiple GPUs on each node of a cluster. GPU nodes of Frontera supercomputer each contain 4 NVIDIA Quadro RTX 5000 GPUs. To further improve the performance of the parameter sweeps, we use multiple GPUs on one GPU node. In this approach, we split the parameter combinations into equal-size subsets and assign each subset to one GPU. Multiple GPUs then execute simulations with the designated parameter combinations concurrently. We investigate the performance of the GPU implementation of the CALM model with both shared memory and DRAM using 2 and 4 GPUs of one GPU node on Frontera. Table 3 presents the runtimes of these four tested scenarios. These results show good speedup when we use multiple GPUs concurrently.

4.8 Hybrid Implementation

We now describe a hybrid CPU–GPU implementation in which we use CPU and GPU simultaneously to perform the parameter sweep. Load balancing here becomes a little complicated because the low discrepancy sequence used in the parameter sweep has some regular patterns. Its number-theoretic properties need to be considered while assigning simulations to the GPU and CPU, respectively, as explained later.

GPU nodes on Frontera use Intel Xeon E5-2620 v4 CPUs that are slower than the CPUs of the regular compute nodes (Intel Xeon Platinum 8280 CPUs), and we can use only 16 MPI ranks on one node. A parameter sweep of the CALM CPU implementation with $N=2000$ on one of these nodes takes 934.37 s, while a similar parameter sweep with the CALM GPU implementation takes about 257.09 s. The performance of the CALM CPU on Intel Xeon E5-2620 v4 CPUs and CALM GPU on the NVIDIA Quadro RTX 5000 GPUs gives us an insight into the suitable share for each of them from the parameter space.

In other words, given the runtime of CPU and GPU for parameter sweeps with $N=2000$, one can compute the value of x such that the runtime of x simulations on CPU is roughly equal to the runtime of $N-x$ simulations on the GPU. As we

show below, solving this equation for $N=8000$ suggests assigning around 1700 simulations to the CPU and 6300 simulations to the GPU. $x \times 257.09 = (8000 - x) \times 934.37 \Rightarrow x = 1727$, where the time for 2000 simulations of CALM on the Broadwell CPU is 934.37 s.

4.8.1 Partitioning

We partition the parameter space into two non-overlapping subsets. We assign one subset to the CPU implementation and the other to the GPU implementation. We let NCPU and NGPU denote the sizes of the respective subsets.

Conventional partitioning techniques in parallel computing are block, cyclic, and block cyclic. In the block decomposition, we assign the first 1700 parameter combinations to the CPU and the remaining 6300 to the GPU. We also try the reverse order in which we assign the first 6300 parameter combinations to the GPU and the remaining 1700 to the CPU and call it reverse block decomposition. In the cyclic decomposition, we would distribute alternate parameter combinations to the CPU and GPU, respectively. In the block-cyclic decomposition, we would divide the parameter sequence into chunks of some size C and then apply a cyclic decomposition to the chunks. Since N_{CPU} and N_{GPU} are not close to each other, the use of a cyclic decomposition or a conventional block-cyclic decomposition would lead to severe load imbalance. We, therefore, modify the block-cyclic decomposition to suit our needs.

We use unequal block sizes C_1 and C_2 on the CPU and GPU, respectively. At each iteration of the block-cyclic scheme, we assign C_1 simulations to the CPU and C_2 simulations to the GPU. There is an additional complication with the use of a low-discrepancy parameter sweep. The simulation time differs in different parts of the parameter space, and the Scrambled Halton low discrepancy sequence generates parameter combinations that sample the parameter space in a deterministic manner[5]. In this process, it uses some small prime numbers, and partitioning sizes that resonate with these primes can yield poor load balance. This was observed in[5] when load balancing just on the CPU, while we balance the load across the GPU and CPU. Therefore, we pick C_1 and C_2 such that they are relatively prime, and $C_1 + C_2$ is relatively prime to the primes used in the Scrambled Halton sequence. We choose three pairs of numbers with the above characteristics: $(<8, 29>, <19, 64>$, and $<19, 64>)$ to test this modified decomposition.

In each experiment, we assign the first block or chunk to the GPU and the second one to the CPU, and we continue with this order for the modified block-cyclic method. We also repeat the same approaches with the reverse order (assign the first block or chunk to the CPU and the second one to the GPU) to see if that can affect the load balance. Table 4 presents the results of these experiments.

Table 4 shows that certain choices of sizes in the modified decomposition technique decrease load imbalance and the total runtime of the parameter sweep. This demonstrates that considering the number-theoretic features of the low discrepancy sequence can further increase the performance. In particular, the block-cyclic scheme with block sizes equal to 19 and 64 takes 10 s less than the block decomposition. Besides, we observe that the hybrid implementation of the CALM model is approximately 18% better than the best single GPU implementation. The use of multiple GPUs will further increase performance.

4.8.2 Real-Time Simulations

We now consider using the GPU implementation to obtain results within the real-time constraints of a couple of minutes. We use a smaller parameter sweep size to accomplish this, with $N=2000$. We have already shown that it yields qualitatively, accurate results. For this experiment, we use four NVIDIA Quadro RTX 5000 GPU on one GPU node of Frontera supercomputer to run this parameter sweep with the CALM GPU code. Table 5 represents the runtimes of these parameter sweeps on DRAM and shared memory. We observe that in these simulations, the use of shared memory leads to a slightly better performance than with DRAM, in contrast to the results of Table 3. This is caused by the trade-off between the use of shared memory and the L1 cache mentioned in "CALM Implementation on GPU". However, with the smaller parameter sweep of Table 5, a decrease in the use of shared memory increases the available L1 cache, thus not hindering the performance of the shared memory implementation. In both cases, the runtime is less than two minutes. This clearly shows that the CALM GPU can be used with the real-time constraints of decision-making meetings. We are currently using this model for a more accurate analysis of the risk of COVID-19 spread in airplanes than would not have been possible earlier.

In addition, as the results of Table 3 show, we can run a parameter sweep with N = 8000, which produces quantitatively accurate results in about

Table 4: Run-Time (in seconds) of the parameter sweeps of the $CALM_{Hybrid}$ model with $N=8000$.

Decomposition	GPU time	CPU time	Total time
Block	800.95	800.75	801.83
Reverse block	771.17	794.79	795.95
Block-cyclic<8, 29>	796.22	790.23	797.25
Reverse block-cyclic<8, 29>	791.50	790.04	792.70
Block-cyclic<16, 57>	792.47	788.21	793.28
Reverse block-cyclic<16, 57>	792.55	790.72	793.75
Block-cyclic<19, 64>	790.12	790.91	791.93
Reverse block-cyclic<19, 64>	790.25	789.57	790.98

Table 5: Run-Time of the parameter sweeps of the CALM model on four GPUs with $N=2000$.

Memory type	Run-Time (s)	STDEV
DRAM	113.29	2.10
Shared memory	104.00	1.08

4 min. This progress is a big step toward enabling real-time policy analysis with quantitatively converged parameter sweeps.

5 Validation

One also needs to show that the results from our model match real pedestrian movement to validate the model. Such results, focused on the mechanics of pedestrian motion, are presented in[21]. As an instance, we compared the deplaning times of three different airplanes (Boeing B757-200 with 182 seats, Boeing B757-200 with 201 seats and CRJ-200 with 50 seats) against the empirically observed times. For Boeing B757-200 with 182 seats, the empirically observed deplaning time is in the range of [10.71, 12.13] minutes and the deplaning times of simulations with our model are in the range of [8.21, 16.43] minutes.

6 Application to COVID-19

Public health warnings recommend maintaining a 1.83 m (6 feet) distance to avoid infection through respiratory droplets of an infected person. However, there is evidence of the droplets being carried by airflow[18], in which case a 1.83 m distance may not be sufficient. We wish to examine if an error in this threshold could lead to a much larger risk of infection spread than expected from models that use a 1.83 m threshold. For symmetry, we also consider the possibility of the true threshold being 1.22 or 2.44 m (4 or 8 feet). Figures 2 and 4 show the number of contacts for each threshold, while Table 6 gives the average number of contacts. The contacts are obtained by summing the number of distinct pairs of passengers within the distance threshold every 1.25 s of deplaning.

We see that a 2.44 m threshold leads to only around a 15% increase in contacts over the recommended 1.83 m, while a value of 1.22 m would lead to around a 38% decrease in the number of contacts. This analysis suggests that an error in the threshold would not lead to a substantial increase in risk at the higher end, while it would lead to a substantial decrease in the risk at the lower end. The CALM model enabled this analysis on a single node with 4 GPUs. SPED would have required massive parallelism.

7 Related Work

Work on the SPED model, explained earlier, is most closely related to ours. It has not been ported to GPUs. However, there are other works that relate to pedestrian dynamics on GPUs.

For example, Richmond and Romano[19] discuss a framework for agent-based pedestrian dynamics on GPUs (each pedestrian in our model could be considered an agent). The focus there is on a software framework that could be used to implement pedestrian dynamics on GPUs. New models that are efficient on GPUs are not proposed, unlike in our work. In fact, the focus of the GPU is on visualization.

Dutta et al.[10] propose two pedestrian dynamics models and port them onto GPUs. This is the conventional approach where a model is developed for an application and then optimized on a GPU. They obtained a factor 18 performance improvement *over a single-threaded CPU* code. In contrast, we obtain a greater improvement in performance *over a multi-threaded code* with 56

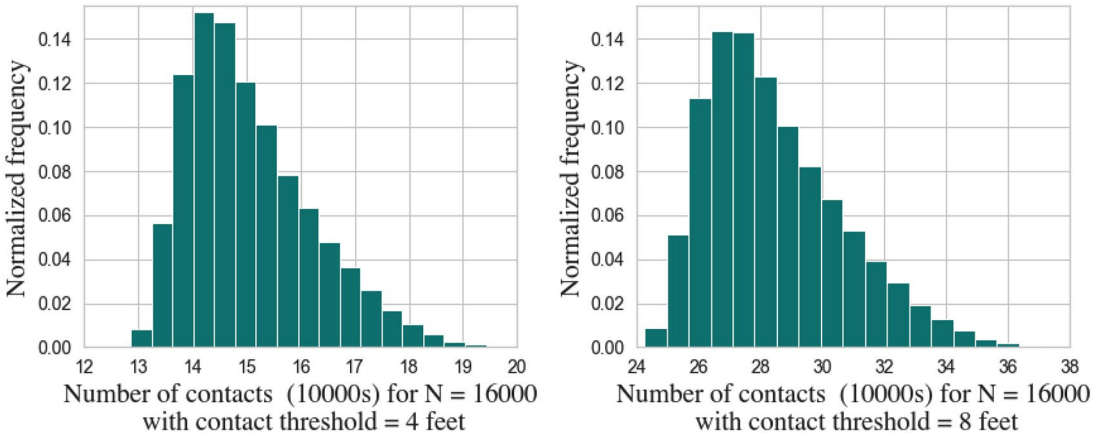


Figure 4: Histograms of the number of contacts with different contact thresholds.

Table 6: Average number of contacts with different contact thresholds for $N = 16{,}000$.

Contact threshold	Average number of contacts
1.22 m (4 feet)	150,587
1.83 m (6 feet)	245,763
2.44 m (8 feet)	284,460

threads, even with a single GPU. Was and Mroz[24] too take a conventional approach and port two existing pedestrian dynamics models to GPUs.

Molecular dynamics with short-range forces has some similarity with pedestrian dynamics, and there is much work in porting such codes to GPUs[20]. However, there are some fundamental differences too. For one, the forces in pedestrian dynamics are often not anti-symmetric because pedestrians are influenced more by those in front of them on their path than those around them in other directions. In addition, while molecular dynamics optimizations may also try to develop approximations to the original force expressions to speed up computation, these approximations have to be very accurate, reflecting the physics. Consequently, then cannot take an architecture-driven modeling approach as we have.

8 Conclusions

We developed the idea of creating a human movement model that was, by design, geared to efficient computation on NVIDIA GPUs. We showed that by relaxing the requirement that it replicates an existing model's computation accurately in a single simulation, we could obtain around a factor 125 improvement in performance. Using all GPUs available on a single GPU node in a cluster, we are able to obtain performance that can meet the real-time constraints of decision-support systems for policy-making in emergencies. Our approach can help with computationally efficient simulations of complex systems involving human behavior, which is critical to decision-making in a broader context than our specific example.

In future work, we wish to improve the performance so that real-time constraints are met with quantitatively accurate simulations rather than just qualitatively accurate simulations. That is, using $N = 8000$ or 16,000, rather than 2000. One direction for exploring such performance improvement is to re-examine half-precision floating-point numbers. It may require developing a novel algorithm that lets the small increments be accumulated.

Publisher's Note

Acknowledgements

The authors acknowledge the Texas Advanced Computing Center (TACC) for providing HPC resources. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation or other funding sources.

Author Contributions

MSL and AS designed the work; MSL, TI, and RG conducted the experiment(s); MSL, AS, and RG analyzed the results; and SN collaborated in the design of the model and experiments. All authors reviewed the manuscript.

Funding

This material is based upon work supported by the National Science Foundation under Grants No. 1931511 and 2027514.

Declarations

Conflict of Interest

The authors declare no competing interests.

Received: 25 March 2021 Accepted: 7 June 2021
Published online: 31 July 2021

References

1. Alahi A, Goel K, Ramanathan V, Robicquet A, Fei-Fei L, Savarese S (2016) Social LSTM: Human trajectory prediction in crowded spaces. In Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016. IEEE
2. Brenner D, Harrison J, White C, Colton R (1991) Molecular dynamics simulations of the nanometer-scale mechanical properties of compressed buckminsterfullerene. Thin Solid Films 206:200–223
3. Burstedde CK, Schaddschneider K, Zittartz J (2001) Simulation of pedestrian dynamics using a two-dimensional cellular automaton. Physica A: Stat Mech Appl 295(3):507–525
4. Chen J, Revels J (2016) Robust benchmarking in noisy environments. arXiv preprint
5. Chunduri S, Ghaffari M, Lahijani MS, Srinivasan A, Namilae S (2018) Parallel low discrepancy parameter sweep for public health policy. In 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID), 2018. IEEE
6. Funk S, Salathé M, Jansen VAA (2010) Modelling the influence of human behaviour on the spread of infectious diseases: a review. J R Soc Interface 7:1247–1256
7. Helbing D, Farkas I, Vicsek T (2000) Simulating dynamical features of escape panic. Nature 407(6803):487–490
8. Helbing D, Molnar P (1995) Social force model for pedestrian dynamics. Phys Rev E 51(5):4282
9. Derjany P, Namilae S, Liu D, Srinivasan A (2020) Multiscale model for the optimal design of pedestrian queues to mitigate infectious disease spread. PLoS ONE 15(7):e0235891
10. Dutta SB, McLeod R, Friesen M (2014) GPU accelerated nature inspired methods for modelling large scale bi-directional pedestrian movement. In Proceedings of the International Parallel and Distributed Processing Symposium Workshops. IEEE
11. Henderson LF (1971) The statistics of crowd fluids. Nature 229(229):381–383
12. Joseph EC, Conway S, Sorensen R, Monroe K (2016) An investigation and evaluation of the scientific results from the NCSA Blue Waters supercomputer system. In IDC Special Report to the National Center for Supercomputing Applications (NCSA)
13. Morokoff W, Caflisch R (1994) Quasi-random sequences and their discrepancies. SIAM J Sci Comput 15(6):1251–1279
14. Namilae S, Srinivasan A, Mubayi A, Scotch M, Pahle R (2017) Self-propelled pedestrian dynamics model: application to passenger movement and infection propagation in airplanes. Phys A 465:248–260
15. Namilae S, Derjany P, Mubayi A, Scotch M, Srinivasan A (2017) Multiscale model for pedestrian and infection dynamics during air travel. Phys Rev E 95(5):052320
16. NVIDIA (2018) NVIDIA Turing Architecture. https://www.nvidia.com/content/dam/en-zz/Solutions/design-visualization/technologies/turing-architecture/NVIDIA-Turing-Architecture-Whitepaper.pdf
17. Okazaki S, Matsushita S (1993) A study of simulation model for pedestrian movement with evacuation and queuing. In: Smith RA, Dickie JF (eds) International conference on engineering for crowd safety. Elsevier, Amsterdam, pp 271–280
18. Ong SWX, Tan YK, Chia PY, Lee TH, Ng OT, Wong MSY, Marimuthu K (2020) Air, surface environmental, and personal protective equipment contamination by severe acute respiratory syndrome Coronavirus 2 (SARS-CoV-2) from a symptomatic patient. JAMA 323(16):1610–1612
19. Richmond P, Romano DM (2008) A high performance framework for agent based pedestrian dynamics on GPU hardware. Proc EUROSIS ESM 20:27–29
20. Rodrigues CI, Hardy DJ, Stone JE, Schulten K, Hwu W-MW (2008) GPU acceleration of cutoff pair potentials for molecular modeling applications. In Proceedings of the 5th Conference on Computing Frontiers, pp 273–282
21. Sadeghi Lahijani M, Islam T, Srinivasan A, Namilae S (2020) Constrained linear movement model (CALM): Simulation of passenger movement in airplanes. PLoS ONE 15(3):e0229690
22. Srinivasan A, Sudheer CD, Namilae S (2016) Optimizing massively parallel simulations of infection spread through air-travel for policy analysis. (2016). In Proceedings of the IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID), pp 136–145. IEEE

23. TechPowerUp (2018) NVIDIA Quadro RTX 5000. https://www.techpowerup.com/gpu-specs/quadro-rtx-5000.c3308. Last accessed: 23 Mar 2021
24. Was J, Mroz H (2015) GPGPU computing for microscopic simulations of crowd dynamics. Computing Informatics 34:1418–1434
25. Zębala J, Ciępka P, RezA A (2012) Pedestrian acceleration and speeds. Problems Forensic Sci 91:227–234

Mehran Sadeghi Lahijani is a Ph.D. candidate in the Department of Computer Science at Florida State University. His research spans the fields of high-performance computing, parallel and distributed systems, and algorithms.

Rahulkumar Gayatri is an Application Performance Specialist at NERSC, LBNL. Previously he did his postdoc in the NESAP program at NERSC. He completed his Ph.D. from Barcelona Supercomputing Center, Spain. His research interests include parallel programming frameworks and high-performance computing.

Tasvirul Islam is a Research Assistant at the University of West Florida. He completed his masters in Computer Science from the University of West Florida. His research interests are in high-performance computing for scientific applications and parallel programming.

Ashok Srinivasan is the William Nystul Eminent Scholar Chair and Professor at the University of West Florida and a Fulbright Fellow. He obtained his Ph.D. in Computer Science from the University of California, Santa Barbara (UCSB). His research interests lie in the application of supercomputing to scientific and public health policy applications.

Sirish Namilae is Associate Professor of Aerospace Engineering at Embry-Riddle Aeronautical University. He is an alumnus of the Indian Institute of Science, where he did ME in Materials Science, and he has a Ph.D. in Mechanical Engineering from Florida State University. His research interests are in the areas of particle dynamics, multiscale modeling, and composite materials.