Optimizing Parallel Particle Tracking in Brownian Motion Using Machine Learning

Journal Title
XX(X):1–16

DOI: 10.1177/ToBeAssigned
www.sagepub.com/
SAGE

Srđan Nikolić[1], Nenad Stevanović[1], Miloš Ivanović[1]

Abstract

In this paper, we present a generic, scalable and adaptive load balancing parallel Lagrangian particle tracking approach in Wiener type processes such as Brownian motion. The approach is particularly suitable in problems involving particles with highly variable computation time, like deposition on boundaries that may include decay, when particle lifetime obeys exponential distribution. At first glance, Lagranginan tracking is highly suitable for a distributed programming model due to the independence of motion of separate particles. However, the commonly employed **Decomposition Per Particle** (DPP) method, where each process is in charge of a certain number of particles, actually displays poor parallel efficiency due to the high particle lifetime variability when dealing with a wide set of deposition problems that optionally include decay. The proposed method removes DPP defects and brings a novel approach to discrete particle tracking. The algorithm introduces master/slave model dubbed **Partial Trajectory Decomposition** (PTD), in which a certain number of processes produce partial trajectories and put them into the shared queue, while the remaining processes simulate actual particle motion using previously generated partial trajectories. Our approach also introduces meta-heuristics for determining the optimal values of partial trajectory length, chunk size and the number of processes acting as producers/consumers, for the given total number of participating processes (**Optimized Partial Trajectory Decomposition, OPTD**). The optimization process employs a surrogate model to estimate the simulation time. The surrogate is based on historical data and uses a coupled machine learning model, consisting of classification and regression phases. OPTD was implemented in C, using standard MPI for message passing and benchmarked on a model of ^{220}Rn progeny in the diffusion chamber, where particle motion is characterized by an exponential lifetime distribution and Maxwell velocity distribution. The speedup improvement of OPTD is approximatelly 320% over standard DPP, reaching almost ideal speedup on up to 256 CPUs.

Keywords

Lagrangian particle tracking, Distributed computing, Evolutionary algorithms, Machine Learning, MPI

Introduction

Random motion of particles suspended in liquid or a gas is described by Brownian motion (Wiener process) and can be modelled by means of Lagrangian method. The simulation of a large number of particles can be time-consuming, due to frequent motion changes caused by collisions with other molecules of liquid or gas. Reducing computational time is of particular interest, bearing in mind that the results could provide support in decision-making when the dynamics of harmful substances is being monitored. Special attention should be given to simulations in which different particles require different computation time to generate their trajectories until reaching decay or deposition.

Generating a trajectory of a single particle is independent from generating other particles' trajectories, which at the first glance makes this task trivial to parallelize by a simple static decomposition. By using **Decomposition Per Particle** (DPP), each processor is responsible for modeling trajectories of approximately equal number of particles, as demonstrated by Lenôtre (2016), Roberti et al. (2005) and Larson and Nasstrom (2002). When solving problems

[1] University of Kragujevac, Faculty of Science, Kragujevac, Serbia

Corresponding author:
Miloš Ivanović, University of Kragujevac, Faculty of Science, R. Domanovića Str. 12, 34000 Kragujevac, Serbia.
Email: mivanovic@kg.ac.rs

in which the particle terminates its motion by decay or deposition, this approach shows the downside in reduced parallel efficiency caused by variability of computation time necessary to generate trajectories of individual particles. Besides this method there is a domain decomposition approach, where space is partitioned into subdomains and each subdomain is addressed to a different processor. With this kind of decomposition, there is an additional communication overhead tied to particles transferring from one subdomain into another, as showed by Charles et al. (2008) and Beaudoin et al. (2007).

In this paper, we present a novel approach to the parallelization of the Lagrangian particle tracking, applicable in Wiener type processes characterized by stationary independent increments. The algorithm eliminates the disadvantages of DPP by introducing medium-grain parallelism that is embodied in a novel **Partial Trajectory Decomposition** (PTD) concept. The main idea behind the PTD approach is that multiple processors should be responsible for creating behaviour history of a single particle. Since multiple processors can be involved in the simulation of a single particle, we break the problem domain into smaller pieces compared to DPP. The algorithm uses a master-slave parallelization model in which *partial trajectory producers* generate so-called *partial trajectories* with constant length and put them into the *shared queue*. On the other hand, there are *particle simulators* that consume those partial trajectories to run real particle tracking.

Selecting optimal values of the parameters, such as partial trajectory length, chunk size and the fraction of processors acting as *partial trajectory producers*, is very important for achieving maximum efficiency of the method and keep the load balanced. An incorrect choice of these parameters could possibly result in the congestion of the *shared queue*. Since determining the optimal values of PTD parameters is not a straightforward (linear) task, we employed the evolutionary-based meta-heuristics. To assess the simulation time for any combination of the PTD parameter values, the evolutionary optimization uses an approximate machine learning model based on historical records on previous runs. The improved method that makes use of the evolutionary algorithm and surrogate model is referred to as **Optimized Partial Trajectory Decomposition** (OPTD) henceforth.

We demonstrate the performance gain for the case of tracking Radon progeny in a diffusion chamber. It is a good example of a time-consuming simulation, where individual particles have approximately exponential distribution of the computation time. More precisely, there is a small chunk of particles that require much more time to generate their trajectories than the majority of the particles in a chamber.

Related work

Lagranginan tracking is an essential technique in numerous research areas, some of which employ Monte-Carlo-based models. In order to speed up the simulation, many of those research studies such as Roberti et al. (2005), Lenôtre (2016), Larson and Nasstrom (2002), Charles et al. (2008), Breuer et al. (2006) and Beaudoin et al. (2007) addressed the parallelization of particle tracking algorithms. In most cases, the authors suggest two strategies of decomposition: *(i)* Domain Decomposition, in which the space integration domain is partitioned into smaller volumes, and each volume (subdomain) is addressed to a different processor as demonstrated by Roberti et al. (2005), and *(ii)* Decomposition Per Particle (DPP), where each processor carries out a certain number of particles throughout their lifetime. Nevertheless, in the first decomposition approach, the additional communication is required when a particle moves from one subspace into another. There is a higher probability of particles moving from one subspace to another with the increase of the number of processors involved. Additionally, in the real-world use cases, the particles would not be distributed equally over subdomains, which would eventually lead to an unequally balanced load. The second decomposition approach does not involve any additional communication burden, as stated above, but if particles are characterized by highly variable computation time, that still brings unequally balanced processor utilization.

Lenôtre (2016), Roberti et al. (2005) and Larson and Nasstrom (2002) introduce a parallel approach based on decomposition per particle (DPP) to avoid any additional communication required in the domain decomposition. To the best of our knowledge, none of them considers the problem of reduced parallel efficiency.

Charles et al. (2008) created a parallel simulation of the sedimentation of a large number of particles in shallow water using the Lagrange method. They present a parallel implementation based on DPP with three different sediment suspension methods. Nonetheless, all three approaches in their research are implemented on the assumption that there was a large number of particles. In that case, there is a little chance that all particles for which a specific processor is in charge are suspended and that the processor remains idle.

In Breuer et al. (2006), the authors implemented an idealized mouth-throat model to predict the success of aerosol therapy. The simulation of the model occurs in

two phases, *continuous* which is calculated by means of the finite-volume method and *particle phase* simulated using the Lagrange approach. Both phases were parallelized using the domain decomposition method, with the potential drawback that particle tracking is not balanced among participating processors. However, this defect does not affect the efficiency of the entire algorithm to a great extent, because the continuous phase requires much more processing power than the particle phase.

Beaudoin et al. (2007) describe a parallel algorithm for the simulation of the soluble substances within highly heterogeneous porous substances. The algorithm relies on the independence of particles and the domain decomposition ends when all particles dissolve or leave the monitored area. They emphasize that the area in which the particles are monitored must be as large as possible, so that particles would not jump out. It is also underlined that it is generally difficult to calculate the computational power required to perform the simulation and that the two main factors affecting the parallel algorithm are heterogeneity and particle diffusion. ~~The presented speedup indicates the problem which we attempt to overcome by our own approach.~~ However, Stochastic Differential Equation (SDE) considered by Beaudoin et al. (2007) contains the fluid component besides the Brownian motion component. Unfortunately, our (O)PTD approach cannot be applied to this kind of problem. For our decomposition to work, we assume a fully stochastic process with stationary independent increments (Wiener process).

To the best of our knowledge, none of the above methods, whether using static or dynamic domain decomposition or particle decomposition, is highly scalable independently of the number of CPUs and the number of particles involved. The problem can be rectified to a certain extent by employing dynamic particle decomposition, which calls for additional communication, thus further slowing down the execution. The major issue that has not been treated in the above mentioned studies arises when processors outnumber particles to track, i.e. at the very end of the deposition/decay process. As a result, increasing number of processors will actually lead to significant slow down. With our approach, the tracking speedup is apparent even with only a single particle remained to track. Nonetheless, there is no need to modify the decomposition during the simulation, if we assume that the number of processors involved is constant.

The progeny of ^{220}Rn in the diffusion chamber

As a representative benchmark of our (O)PTD method, we have chosen to track ^{222}Rn progeny in the diffusion chamber, being a pure Wiener process and representing a large class of particle tracking problems in research and engineering.

The diffusion chamber shown in Fig. 1 is a cylindrically shaped device covered by permeable filter with solid state nuclear track detector placed inside. Radon, which is presumably homogeneously distributed inside the chamber, decays, forming new short-lived progeny atoms. The formed progeny diffuses through the chamber and can further decay or deposit onto the chamber wall. A certain portion of the progeny decay in the air (hereafter referred to as the *air fraction*) and others decay after the deposition onto the wall (hereafter referred to as the *deposited fraction*). The problem setup is identical to the setup described by Nikezić and Stevanović (2005) and Nikezic and Stevanovic (2007).

The progeny particles move randomly, as a result of continuous bombardment from molecules of the surrounding gas (in our case air). Such random motion is known as Brownian motion or Wiener process. The random motion of particles is characterized by velocity direction and magnitude, as well as the path length between two subsequent collisions. All these variables are random in nature.

As stated above, radon is distributed homogeneously in the air chamber. The ^{218}Po, as the first progeny, appears upon Radon decay. Due to the homogeneous distribution, the location of the point where ^{218}Po forms could be generated as proposed by Nikezic and Stevanovic (2007):

$$\begin{aligned} z_0 &= H \cdot u_1, \\ r &= R \cdot \sqrt{u_2}, \\ \varphi &= 2\pi \cdot u_3, \end{aligned} \tag{1}$$

where $u_i (i \in \mathbb{N})$ are uniform random numbers between 0 and 1. Consequently

$$\begin{aligned} x_0 &= r \cdot cos\varphi, \\ y_0 &= r \cdot sin\varphi. \end{aligned}$$

Random sampling of each ^{218}Po lifetime, T, is given as

$$T = -\tau \ln(u_4), \tag{2}$$

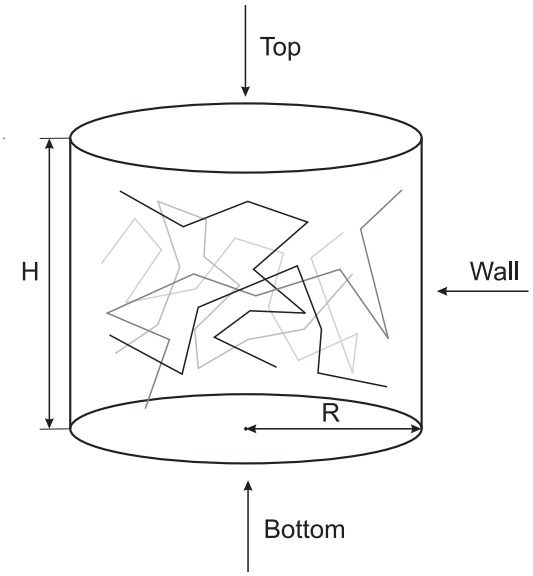


Figure 1. Diffusion chamber detector

where $\tau = T_{\frac{1}{2}} \ln 2$ denotes the mean lifetime. A random unit vector (p_x, p_y, p_z) represents the direction of motion of a progeny atom, while the distance λ, up to the collision with another atom, is taken from the exponential distribution as

$$\lambda = -l \cdot \ln(u_5), \tag{3}$$

where l designates the mean free path. The speed of the progeny atom v could be determined by Maxwell distribution. The time until collision is $\frac{\lambda}{v}$, giving a particle state for a Monte-Carlo step $i+1$ as

$$\begin{aligned} x_{i+1} &= x_i + \lambda \cdot p_x, \\ y_{i+1} &= y_i + \lambda \cdot p_y, \\ z_{i+1} &= z_i + \lambda \cdot p_z, \\ t_{i+1} &= t_i + \frac{\lambda}{v}. \end{aligned} \tag{4}$$

A ^{218}Po atom decays if all collisions remain within the chamber throughout its entire lifetime. Otherwise, the progeny deposits onto the chamber wall. In our model, any particle contact with the chamber wall means immediate deposition.

In the previous paragraphs we have described the direct simulation of the random motion of the particles in the air volume inside a diffusion chamber. However, this simulation is immensely time-consuming because a particle experiences about 10^9 collisions per second and moves randomly at very short distances. The creation of a single particle history takes more than several hours on Intel Core-i7 based computers. Furthermore, in order to calculate the air deposited fractions or the distribution of the deposited progeny, thousands of particle histories are needed. In addition, the computational time distribution is highly variable. More precisely, more than 90% particles deposit at the very beginning of the simulation, while a relatively small fraction of particles require much longer computation time until reaching deposition on the wall or decay in the air. If we take a coarse-grained parallelization approach assigning each processor a certain amount of particles (DPP), this heterogeneity will be charged by significant loss of efficiency. One of the key parts of this research is the creation of a finer-grain decomposition approach that will resolve the mentioned heterogeneity issue.

Partial Trajectory Decomposition (PTD)

At first sight, Lagranginan tracking is suitable for a parallel programming model, since all the particle trajectories are entirely independent. We can easily achieve speedup using DPP, where each processor manages a certain number of particles. Nevertheless, when particles are characterized by highly variable computation time, DPP will lead to imbalanced processor utilization. There is a possibility that a very small number of particles require much more computation time, which results in significantly reduced efficiency.

Our novel **Partial Trajectory Decomposition** (PTD) approach (Fig. 2) uses a master-slave model in which **partial trajectory producers** generate trajectories of specified length and put them into the **shared queue**. They generate partial trajectories according to Eq. (4). On the other hand, there are **particle simulators** which consume partial

trajectories from the *shared queue* to perform real particle tracking.

The main idea of the PTD method is that *particle simulators* are still in charge of simulating motion of a certain set of particles, the same as in DPP approach, but it excludes them from the demanding work of generating particle trajectories. *Particle simulators* actually manage the particle motion by bringing together partial trajectories from the *shared queue*. This approach brings finer granularity, achieving a better load balance. In the edge case when only one particle remains in the chamber, our PTD approach still includes all *partial trajectory producers*. The number of particle trajectory producers and particle simulators depends on the number of processors and the configuration of the computer system used to perform the simulation. More details will be provided in the Section *Optimized Partial Trajectory Decomposition*.

The task of a *partial trajectory producer* is to generate partial trajectories of a specified length. At the very beginning, each *particle simulator* takes the initial location of a particle and tracks it by taking partial trajectories from the queue and adding them up. When a *particle simulator* ends up with a certain particle simulation, it takes the initial position of another particle and so on. The *particle simulator* terminates when all particles are deposited or decayed. Fig. 3 presents the algorithm by which a *particle simulator* tracks a particle.

Step 1. Take a partial trajectory from the queue. It is important to note that a message (Fig. 4) does not contain any intermediate points of the partial trajectory, but only the following:

- *Minimal bounding box* of the partial trajectory, defined by points $A \dots G$. This allows representing large partial trajectories by only 8 points.
- *Translation vector* $\vec{X}$.
- *Partial trajectory time*, which represents the time needed to cross the partial trajectory.
- *Random number generator seed*, which can be used to reconstruct the entire partial trajectory if needed.

The presented message structure aims to reduce the amount of data transferred between *partial trajectory producers* on the left side and *particle simulators* on the right side.

Step 2. When a *particle simulator* takes a message from the queue, it is necessary to translate a minimum bounding box to the current particle position, sticking the partial trajectory to the current particle position. Afterwards, we perform the obligatory check if the bounding box intersects the chamber walls. If there is no intersection (Fig. 5a), we move on to **Step 3**, or else (Fig. 5b) we move on to **Step 5**.

Step 3. In case that the minimum bounding box does not intersect the wall, we must also check if the particle has enough lifetime to fulfil the entire partial trajectory. It is an easy task, since the message that contains the partial trajectory also contains time interval to cross it. If the particle does not have enough lifetime to fulfil the entire partial trajectory, we move on to **Step 5** and determine particle decay position, or otherwise move forward to **Step 4**.

Step 4. When we are assured that the particle fulfils the entire partial trajectory and gets a new position by translating by vector $\vec{X}$ (Fig. 6), upon updating the particle motion time, we can safely move on to the next partial trajectory treatment (**Step 1**).

Step 5. The particle deposited somewhere along the partial trajectory and its final position has to be determined. Since the partial trajectory message does not contain a full path, the *particle simulator* itself has to simulate the particle motion along a partial trajectory until deposition/decay. Owing to the fact that the partial trajectory message contains a random number seed, it is entirely possible for the *particle simulator* to reconstruct the identical path. This is an obvious computational overhead, but the communication is reduced by an order of magnitude.

The parameters affecting performance

The main requirement for any parallel algorithm is to minimize interprocess communication and to maintain the optimal load balance. This is not a trivial task in the case of the PTD approach, due to a number of quantities having arbitrary values. First of all, it is necessary to select the optimal value of partial trajectory length (L_{pt}) and a fraction of the processes to act as *particle simulators*, given the total number of processes. It is obvious that the amount of communication is greater if the L_{pt} is smaller. For example, if there is a large number of *partial trajectory producers* and only a single *particle simulator* and relatively small L_{pt} is chosen, it will give rise to a potential congestion of the *shared queue*. On the other hand, larger L_{pt} will reduce the amount of communication among processes, but the particle simulator will devote more of its resources to the reconstruction of the partial trajectories for the purpose of determining the final deposition locations (*Step 5* in the previous section). In addition, even in case of highly optimal

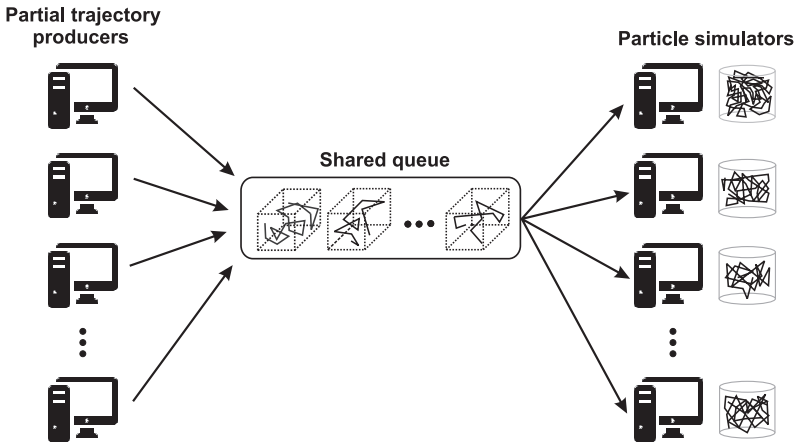


Figure 2. Partial Trajectory Decomposition (PTD) approach to Lagrangian particle tracking

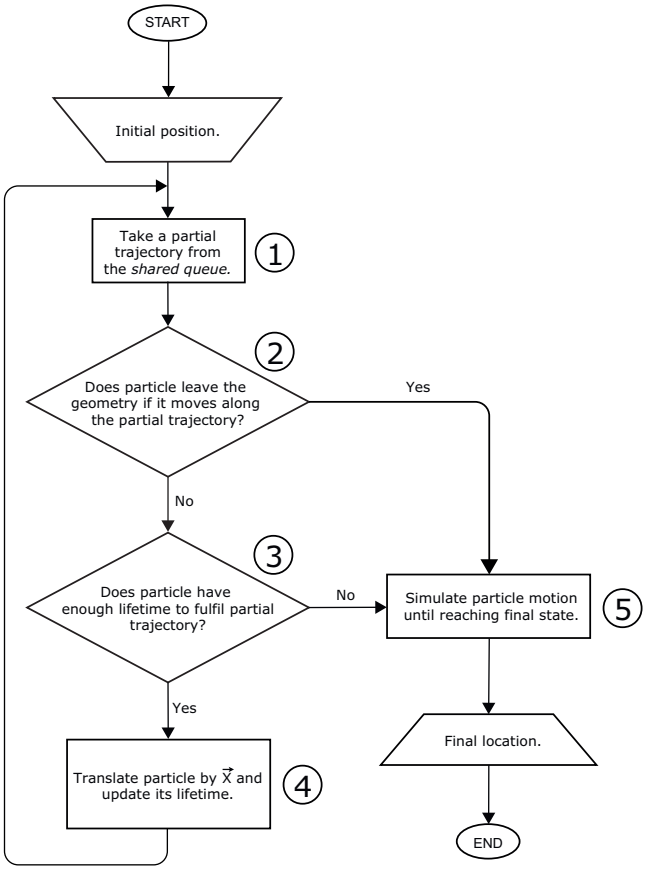


Figure 3. Particle simulator tracking a particle

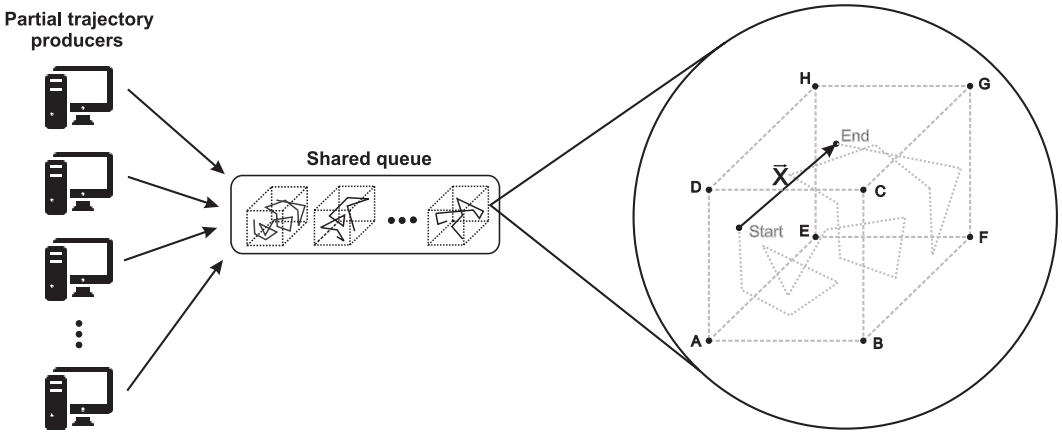


Figure 4. Contents of a message in the *shared queue*

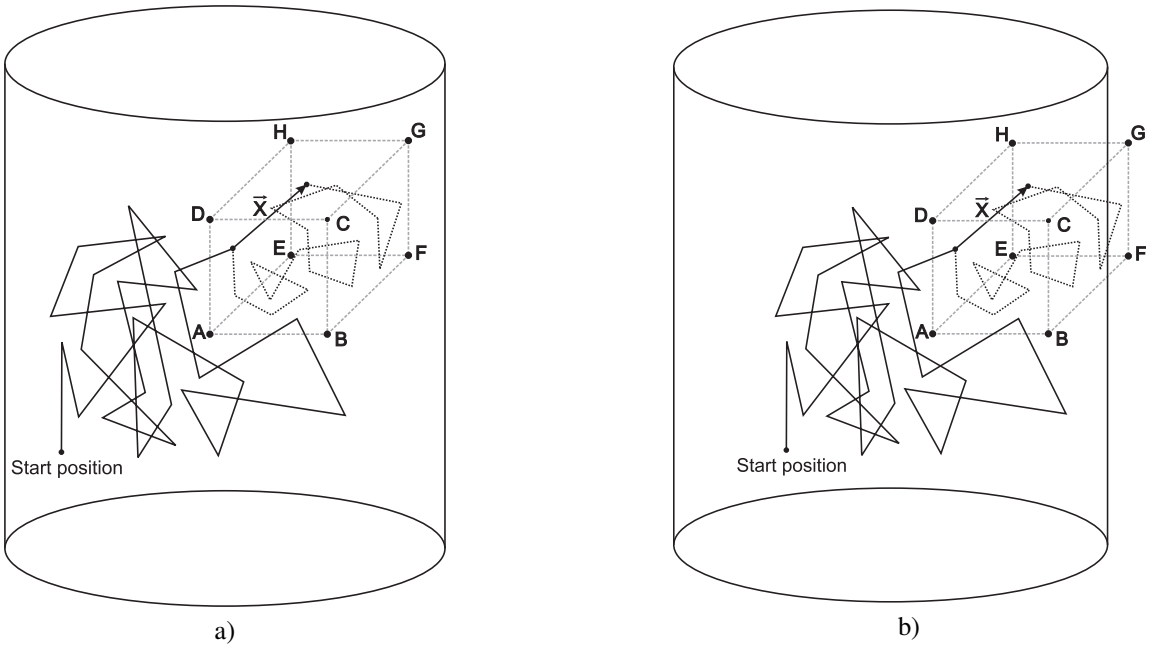


Figure 5. *(a)* The minimum bounding box does not intersect the chamber wall. *(b)* The minimum bounding box intersects the chamber wall.

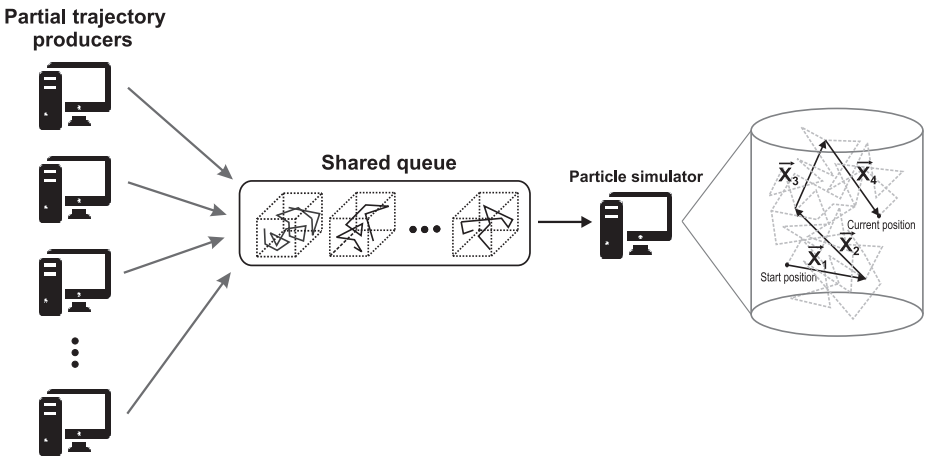


Figure 6. A particle simulator performs real particle tracking by translating current particle position by $\vec{X}$.

parameters, there is still a possibility of congestion due to hardware or software limitations of a HPC system. Network shortages between arbitrary nodes is an obvious example of the efficiency reduction.

To create a robust solution that is also congestion resistant, we had to create a mechanism in which *partial trajectory producers* will produce not only a single partial trajectory at a time, but a chunk of partial trajectories on demand. The optimal trajectory chunk size (N_{chunk}) also depends on the implementation of the PTD and on the specific HPC environment, like all other PTD parameters already introduced.

The MPI implementation of the PTD

We implemented the said approach using standard MPI for message passing on a model of ^{220}Rn progeny in the diffusion chamber (Fig. 7). The process ranked 0 (master) generates initial particle positions and operates the *shared queue* employing non-blocking MPI API. Additional responsibility of the master process is to manage the distribution of the deposited particles.

Each *partial trajectory producer* (N_{prod} processes) generates a partial trajectory chunk on demand and sends it out to the master. The master then sends the request for a new partial trajectory chunk when the previous one is completed. On the right hand side, each *particle simulator* initially sends the request for a particle to the master, and waits for the response. The response includes one of the following: *(i)* the data regarding a particle whose motion is to be simulated, *(ii)* the termination message that all particles in the diffusion chamber are already deposited or decayed. When a particle

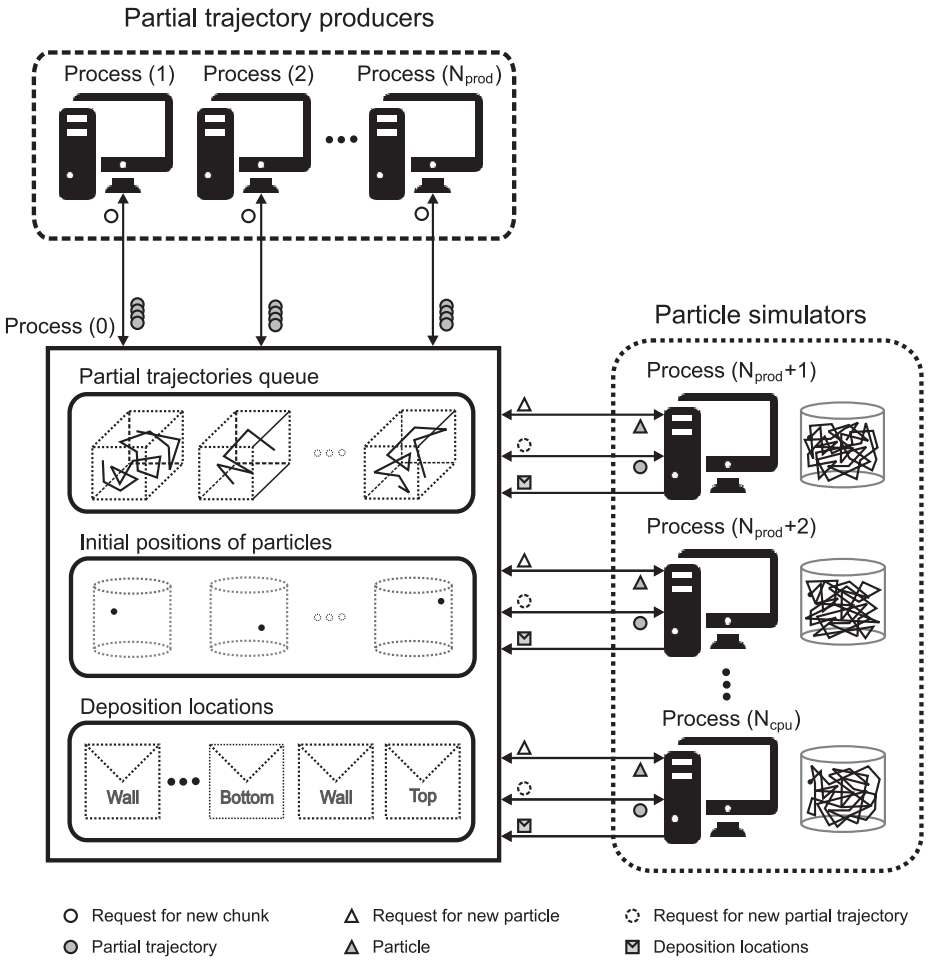


Figure 7. The MPI implementation of the PTD method

deposits, a *particle simulator* sends its final location to the master together with the request for another particle.

For the purpose of generating all randomness in the eqs. (1)-(4) we employed a standard function `drand48_r` that generates pseudo-random numbers using the linear congruential algorithm and 48-bit integer arithmetic. Each of the *partial trajectory producers* has its own seed to ensure the independence of the generated trajectory streams. The seed value is obtained as a sum of the process rank and the Unix timestamp taken from the real-time clock.

The choice of values of the following parameters: *(i)* partial trajectory length (L_{pt}), *(ii)* the number of the *particle simulators* (N_{sim}), and *(iii)* the trajectory chunk size (N_{chunk}), is crucial for the performance. We refer to all these parameters as **PTD parameters** in further text. As mentioned above, their optimal values depend on the total number of CPUs involved and specific HPC computing environment. A wrong selection of PTD parameter values could lead to a total or partial congestion of the master due to its inability to process all incoming requests.

The initial benchmark has shown that changing PTD parameters most certainly affects the total simulation time in a non-linear manner. There are scenarios in which a small change of any PTD parameter causes a profound difference in the simulation time. The major responsibility for this behaviour lies in a total or partial congestion of the master.

Optimized Partial Trajectory Decomposition (OPTD)

In order to improve the performance of PTD and prevent a bad selection of method parameters, especially those that would lead to the total queue congestion, we propose a specific optimization procedure based on evolutionary optimization with a single objective function being *simulation time* T_{sim}. For a given total number of CPUs

$$N_{cpu} = N_{prod} + N_{sim},$$

it searches for the optimal combination of the PTD parameters in the given HPC environment. However, it is impossible to perform the evaluation of the PTD parameter combination analytically since they largely depend on hardware and software configuration. The only absolutely exact way to evaluate the PTD parameter combination is to

execute a real simulation in the target HPC environment, but such a process can be very time consuming and infeasible to complete in a real time frame, since the optimization process requires hundreds of evaluations. Fortunately, it is possible to approximate the simulation time of an individual (PTD parameter combination) requested by the evolutionary optimization by a surrogate model based on machine learning. The surrogate model learns from the acquired historical data to predict the simulation time for various PTD parameter combinations.

As shown in Fig. 8, the surrogate model approximates the simulation time T_{sim} for the given N_{cpu} and various combinations of PTD parameter values suggested by the evolutionary algorithm. The proposed optimization scheme is named Optimized Partial Trajectory Decomposition (OPTD). The OPTD adds one more application layer on top of the PTD stack, which frees the end-user from configuring PTD parameters. The entire system is automated so that the end user is only required to set a total number of available processors (N_{cpu}). Optionally, one is also allowed to modify evolutionary algorithm variables such as the number of individuals, number of generations, mutation probability, etc.

Predicting the simulation time using the surrogate model

The historical data contains simulation time for each historical PTD parameter value combination in the form:

$$(N_{cpu}, L_{pt}, N_{sim}, N_{chunk}, T_{sim}).$$

It became apparent that creating a single regression model for predicting the simulation time over historical data was not a trivial task. The reason for this lies in those samples that caused total congestion, having too high T_{sim} to be covered by a single regression model. Consequently, to achieve acceptable accuracy, we created a surrogate model that actually consists of two coupled ML models (Fig. 9).

The first model is a **classification model** that employs well-known J48 decision tree algorithm described by Hall et al. (2009) over the entire historical data set. It evaluates whether the PTD parameter value combination causes the queue congestion. Its input of historical data was slightly modified by replacing the column T_{sim} by the corresponding binary feature *SQoverflow*. It classifies each PTD parameter combination into *Yes*, if it causes the congestion, or *No* otherwise. During an inference, if the classification model concludes that specific PTD parameter combination causes congestion, such response goes directly to the output. Otherwise, we invoke a **regression model** to predict the simulation time. We have achieved the best accuracy for the regression model by employing an artificial neural network (ANN), which was trained using the same historical data excluding those samples that caused total congestion.

Preventing spurious minima

During evolutionary optimization, we evaluate numerous PTD parameter combinations by the surrogate model. Regardless of the model accuracy, there is a possibility that its suggested total T_{sim} is lower than achievable in practice Jin (2005) due to potential overfitting. It is mandatory to ignore such predictions, but as shown by Jin (2005), there is no single kind of approach to this problem. We have chosen to create a solution which is adapted to the specific problem under consideration by approximating the *ideal simulation time* for a given N_{cpu}. In case that the surrogate predicts T_{sim} less than the ideal simulation time, we judge that individual with the worst fitness value.

The model that estimates the *ideal simulation time* for any N_{cpu} uses data from historical runs as all previous models, but is much simpler. First we determine the minimum historical T_{sim} for each N_{cpu} contained in the data set. A simple multi linear interpolation shown in Fig. 10 estimates the ideal simulation time for any N_{cpu} in range.

The unknown ideal simulation time t_x for a given $N_{cpu} = n$ can be calculated by:

$$\frac{t_x - t_1}{n - n_1} = \frac{t_1 - t_2}{n_1 - n_2}. \tag{5}$$

Results and discussion

We benchmarked the proposed PTD method tracking ^{220}Rn progeny in a cylindrical detector described in Section *The progeny of ^{220}Rn in the diffusion chamber*. First of all, we present the comparison between DPP and PTD methods to demonstrate the influence of the finer-grain parallelism. The second goal of the study was to create a framework that adds an optimization layer (OPTD) with a purpose to obtain the maximum efficiency of the PTD method. Consequently, the second benchmark part represents a comprehensive performance analysis of all considered methods - DPP, PTD, and OPTD.

The diameter of the diffusion chamber was $R = 0.04m$, while the chamber height was $H = 0.08m$. The mean free path in the air has a value of $0.066\mu m$, and a half-life of

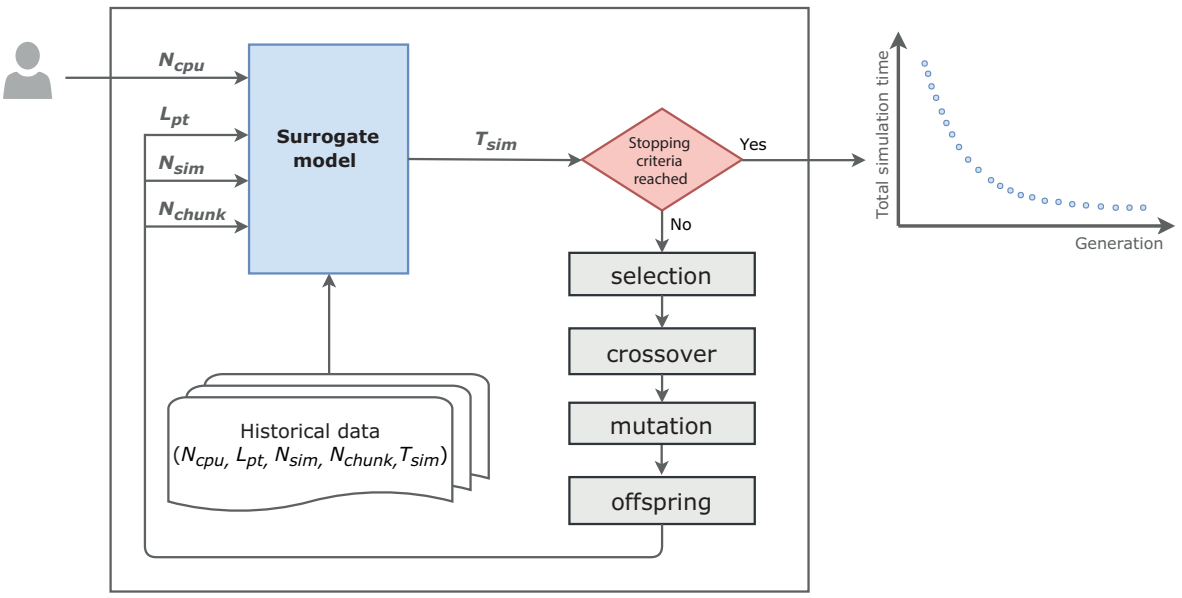


Figure 8. Predicting optimal values of PTD parameters using evolutionary algorithm

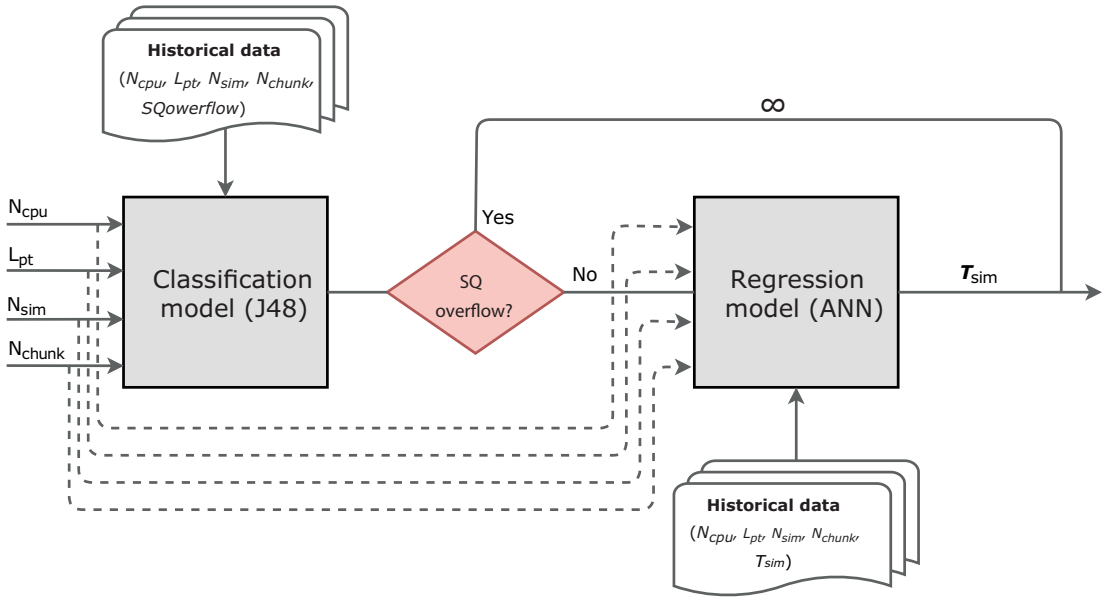


Figure 9. Coupled surrogate model for predicting T_{sim}

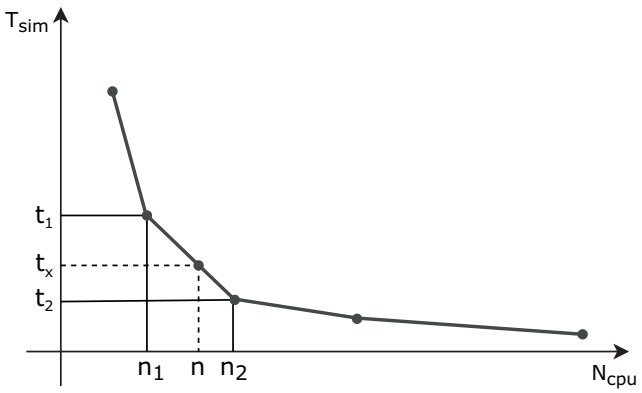


Figure 10. Interpolating *ideal simulation time* from the historical data. The points t_1 and t_2 are known simulation times for a $N_{cpu} = n_1$ and $N_{cpu} = n_2$ respectively.

^{222}Rn is $183s$. We sampled the velocity of ^{218}Po from the Maxwell distribution at normal atmospheric conditions ($293K$ and $1013mbar$).

We carried out testing on the cluster containing 22 worker nodes, each equipped with dual Intel Xeon E5-2670 @ 2.60GHz (16 cores per node) with Infiniband QDR interconnection. The MPI implementation of OPTD approach is availiable on GitHub *.

Decomposition Per Particle (DPP)

The Fig. 11 shows distribution of the simulation time required to deposit/decay each of 1024 particles of Radon progeny on 256 CPUs using standard DPP method. As the histogram suggests, there is a very small number of particles that require much longer computational time to deposit or decay, compared to the majority.

Even 92% of the particles deposited at the very beginning, largely affecting the number of processors involved in simulation over time and leading to extremely low efficiency of 0.29. The efficiency of DPP method even decreases with the increase of the number of processors, as shown in Fig. 12.

Partial Trajectory Decomposition (PTD)

The Fig. 13 presents the speedup of the simulation of ^{220}Rn progeny obtained by PTD and DPP. The PTD parameters were selected manually in three distinct combinations:

- $L_{pt} = 10000, N_{chunk} = 10000, N_{sim} = 1$,
- $L_{pt} = 10000, N_{chunk} = 10000, N_{sim} = 2$,
- $L_{pt} = 10000, N_{chunk} = 10000, N_{sim} = 3$.

It is evident that PTD shows significantly better scalability than DPP in all three benchmarked parameter combinations. However, it is also obvious that it is not an easy task to establish an unambiguous analytical relation for parameter tuning by provisional measurement. It seems that a larger N_{cpu} requires a larger number of *particle simulators* to reach better scalability, although the speedup results are still far from ideal.

Parameter discussion In the identical setup as presented above, we have benchmarked our PTD method in the simulation of 1024 particles of Radon progeny using various N_{cpu} (16, 32, 64, 128, 196 and 256) and various values of PTD parameters:

- L_{pt} : 1000, 10000, 100000, 200000, 300000, 400000 and 500000,
- N_{chunk} : 256, 1024, 4096, 16384, 32768, 65536, 130762 and 261344,
- N_{sim} : 1, 2, 3, 4 and 5,

which gave us the simulation time (T_{sim}) for 1680 different combinations of the PTD parameters and N_{cpu}.

The Pearson correlation between N_{cpu} and T_{sim} on the entire historical data set was only -0.12. This unexpectedly low value was a consequence of (not so rare) occurrence of congestion, which was confirmed by a stronger Pearson correlation (-0.72) between same two variables over historical data that excluded rows with apparent congestion. Table 1 shows how PTD parameters impact T_{sim}. As expected, an increase of the number of processors has the greatest effect on reducing the simulation time. It is also apparent that with a lower N_{cpu}, there is no need for additional *particle simulators*. What should be kept in mind is that even the combination of PTD parameters seemingly optimal for a lower N_{cpu} could lead to a partial or total congestion when a simulation runs on a higher N_{cpu}. Moreover, the changes of the PTD parameters cause a non-linear change of T_{sim} (see Fig. 14). Some values, especially those with congestion, are much higher than 200s, but we cut them off for the sake of better visibility.

Table 1. The best combinations of the PTD parameters obtained from the simulation of 1024 particles of ^{220}Rn progeny for various N_{cpu}.

N_{cpu}	L_{pt}	N_{chunk}	N_{sim}	$T_{sim}[s]$
16	100000	262144	1	138.0
32	100000	262144	1	61.0
64	10000	65536	1	31.5
128	100000	262144	2	16.6
192	100000	130672	2	11.2
256	100000	65536	5	8.2

As illustrated in Fig. 14, with higher L_{pt}, the simulation time reduces with increased number of *particle simulators* N_{sim}. This result is expected, because with a higher value of L_{pt}, with a larger number of processors, a *particle simulator* spends significantly more time on trajectory reconstructions, resulting in a partial congestion of the *shared queue*. A larger N_{sim} obviously reduces the partial congestion as the trajectory reconstructions run in parallel, thus speeding up the execution.

*https://github.com/srdjan034/optd

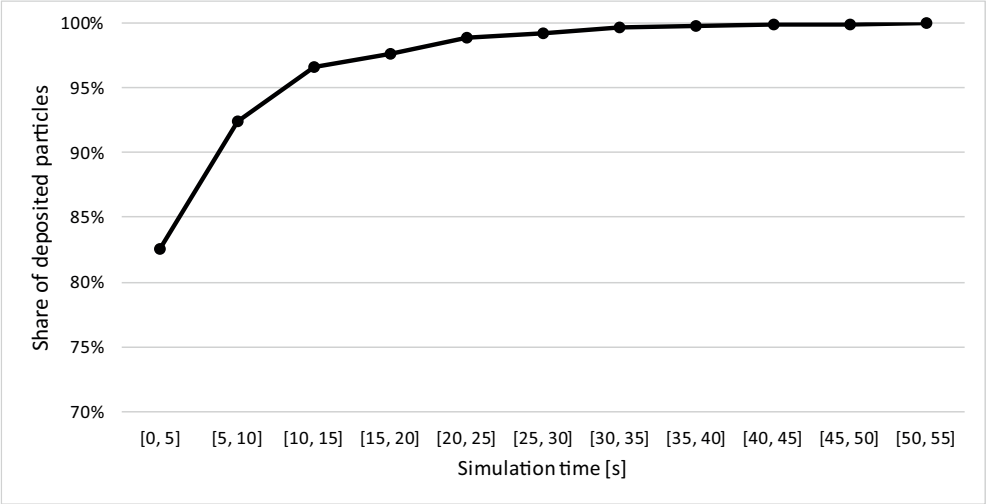
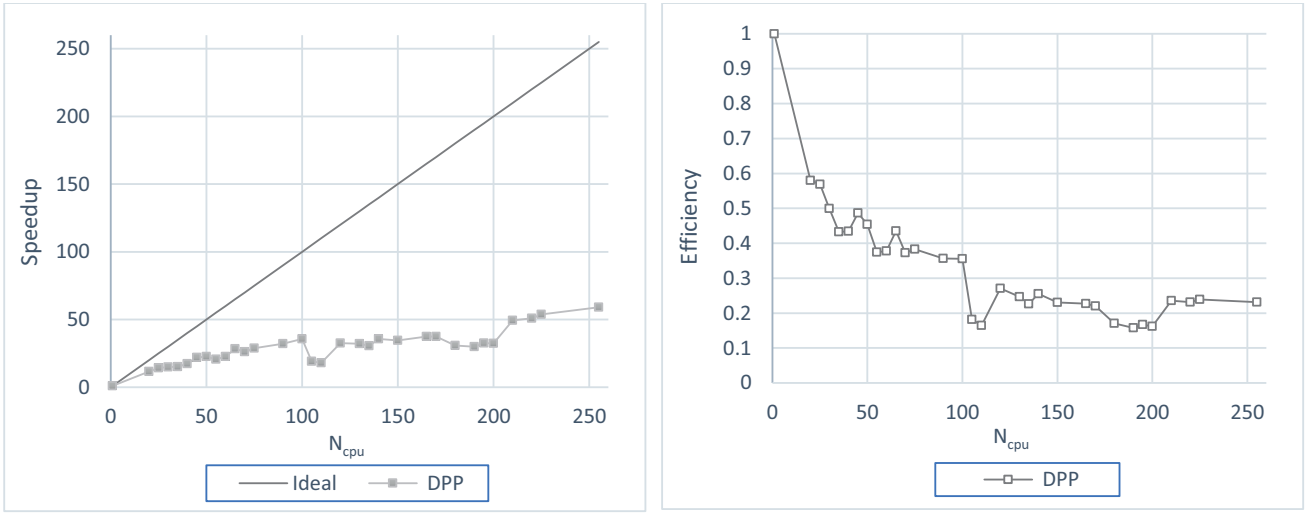


Figure 11. DPP method: The distribution of the simulation time for 1024 particles on 256 CPUs



a) b)

Figure 12. a) DPP speedup. **b)** DPP efficiency.

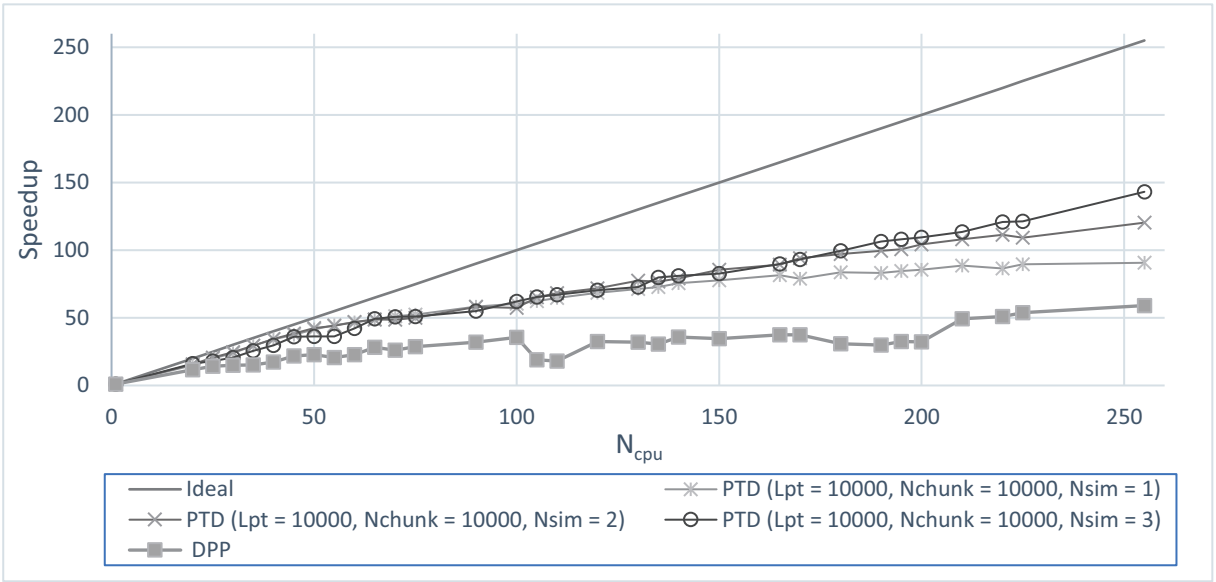


Figure 13. Speedup of PTD with three different parameter combinations in comparison to DPP.

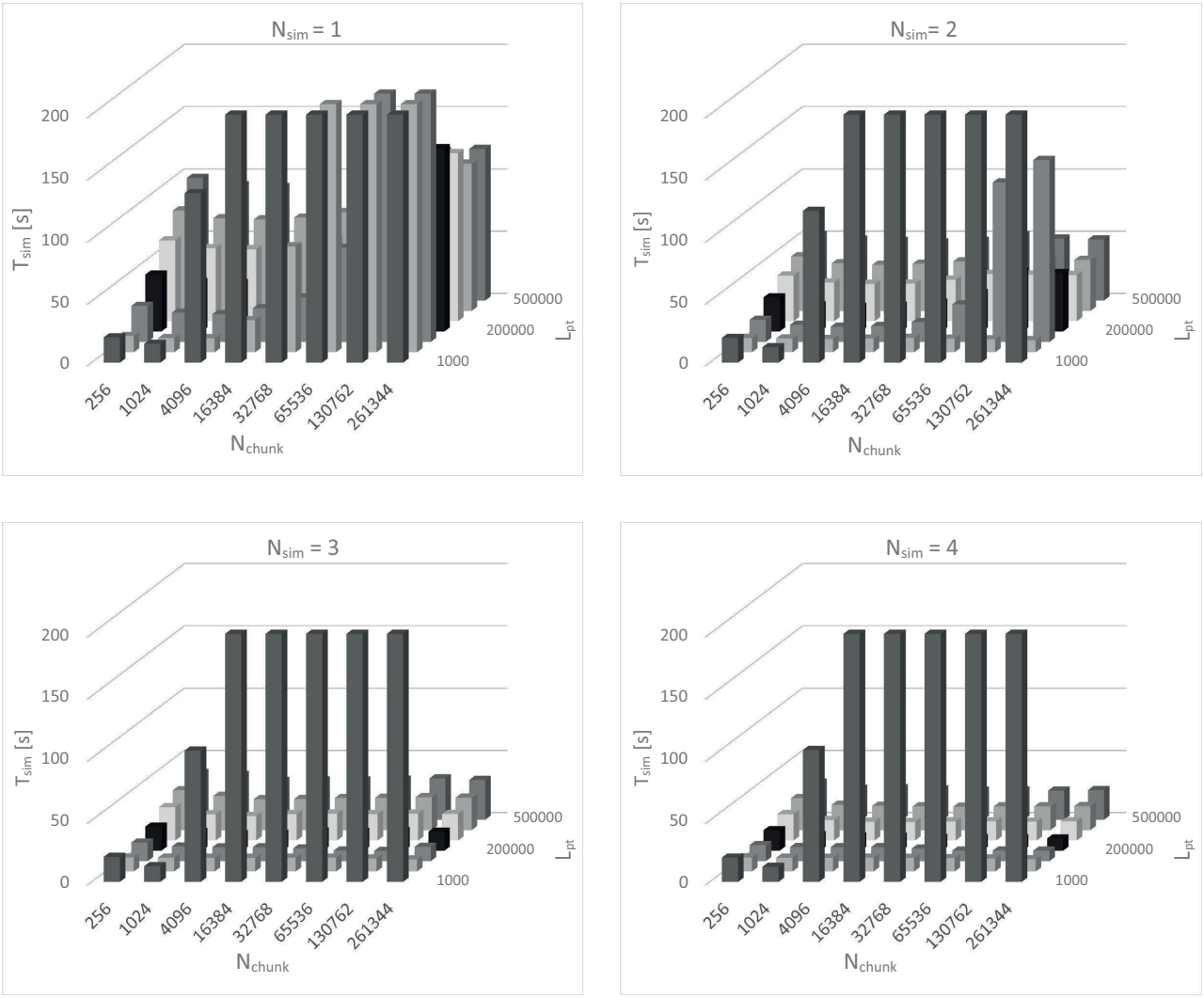


Figure 14. T_{sim} as a function of the PTD parameters in the simulation of 1024 particles of ^{220}Rn progeny on 256 CPUs.

Predicting the simulation time and the occurrence of congestion

The surrogate model for predicting T_{sim} for the given PTD parameters consists of two separate ML models (see Fig. 9) over the historical data described in Section *Parameter discussion*. The actual implementation employs Weka framework developed by Hall et al. (2009) for both models over the same input vector $x = (N_{cpu}, L_{pt}, N_{sim}, N_{chunk})$.

The first model is the classification model created by C4.5 decision tree algorithm developed by Salzberg (1994). As explained above, it classifies the PTD parameter combination into a set *Yes, No*, depending on whether they cause the congestion. In the entire historical data set, 1131 samples were marked as *No* (67.32%), while 549 were marked as *Yes* (32.67%). 10-fold cross validation showed the best performance for the tree having 53 leaves and total tree size of 105. This model correctly classified 1602 (of 1680) instances, giving the satisfactory precision of 95.35%. Table 2 shows the confusion matrix of the classification model. The performance was satisfactory since only 27 samples which actually led to congestion, were classified incorrectly.

Table 2. Confusion matrix of the classification model

	Predicted: No	**Predicted: Yes**
Actual: No	1260	22
Actual: Yes	27	371

The second model is the regression model created using the Multilayer perception (MLP) method. As stated above, the model was created over a subset of 1131 samples that did not cause congestion of the *shared queue*. Its task was to predict the simulation time of 1024 particles of ^{220}Rn progeny for the given PTD parameters and N_{cpu}.

The neural network contained four neurons in the input layer, one neuron in the output layer, and two hidden layers of 20 neurons, all with sigmoid activation. An optimal

number of hidden layers and the number of neurons per layer were tuned manually. We used 10-fold cross validation for the regression model as well, with Root Mean Square Error (RMSE) as an error indicator. The RMSE of the final model was only 2.65s ($\approx 1.5\%$ of the average T_{sim}) with satisfactory Pearson correlation of 0.997. The standard deviation of the folds' RMSE was approximately 2%, showing sufficient suitability of the data set, as well as the adopted regression model.

We also considered excluding N_{chunk} and L_{pt} from the model since Pearson correlation analysis against T_{sim} ranked N_{cpu} and N_{sim} high (>0.5), while N_{chunk} and L_{pt} had negligible values. However, if we take only samples where $N_{sim} = 1$, the Pearson correlation of N_{chunk} and L_{pt} increases to over 0.4, leading us to keep all PTD parameters as attributes.

As mentioned above, we created a coupled surrogate model upon a failure to develop a regression model without any classification preprocessing, over the entire historical data set. For the purpose of creating such a regression model, we benchmarked several ML algorithms, obtaining the best performance from the Random Forest. For the sake of comparison, the best Random Forest regression model had RMSE of as much as 644.68, which was significantly worse than adopted coupled model.

Optimized Partial Trajectory Decomposition (OPTD) performance

In order to benchmark our novel OPTD approach, we have employed the evolutionary optimization with elitism enabled, using MOEA optimization framework by Hadka (2016), to determine the optimal PTD parameters (see Fig. 8). The coupled surrogate model was used to evaluate various combinations of PTD parameters. The number of evaluations was limited to 5000 with the population size of 100 and mutation rate of 0.5. Fig. 15 shows the speedup of the simulation of 1024 particles of ^{220}Rn progeny by OPTD versus PTD and DPP.

It is apparent that OPTD showed significantly better scalability compared to the manually tuned PTD and especially coarse grained DPP. The speedup improvement of OPTD was more than 320% over DPP and 150% over pure PTD, reaching almost the ideal speedup on up to 256 CPUs. Based on the optimal PTD parameter values for a given N_{cpu}, we can also draw the following conclusions:

- As N_{cpu} increases, the optimal trajectory length L_{pt} also increases. In particular, increasing L_{pt} increases the throughput in the sense that more generated collisions can pass through the system using the same amount of communication.
- With the increase of N_{cpu} the number of optimal N_{sim} also increases. This result is expected because a larger N_{sim} speeds up trajectory reconstructions, which prevents the shared queue congestion.
- With the increase of N_{cpu}, the optimal N_{chunk} also increases. This kind of system behavior is logical because, with more CPUs, N_{sim} also increases, preventing shared queue congestion and allowing generating more partial trajectories per single request.
- Let's take the subset where N_{sim} is constant, i.e. $N_{sim} = 1$. In such subset, as N_{cpu} increases, the optimal L_{pt} decreases. The system aims to reduce the path reconstruction burden, as the number of partial trajectories to be processed increases.

The Pearson correlation diagrams that demonstrate the above claims are shown in Fig. 16.

Conclusion

In this paper, we have presented a novel approach in the parallelization of discrete tracking of the chaotic motion of particles through a liquid or gas, within the predefined geometry using the Lagrange method. The behavior of the particles is described by Brownian motion. The emphasis is placed on the specific type of particle simulations characterized by inherent different computational time per particle due to either wall deposition or decay. The presented algorithm removes defects of Deposition Per Particle and brings a finer-grain approach to discrete particle tracking. The main part of the new algorithm is the Partial Trajectory Decomposition, which introduces the master/slave model where multiple workers are responsible for managing tracking of a single particle. Since multiple workers can be involved in the simulation of a single particle, we break the domain decomposition into smaller pieces, which gives us a better load balance compared to the DPP method. To select optimal values of the computational parameters and achieve maximum efficiency, we added the optimization layer to create Optimized Partial Trajectory Decomposition (OPTD). The OPTD approach contains a mechanism that decides upon the optimal values of relevant PTD parameters. That mechanism is based on the evolutionary algorithm, which itself employs the surrogate model to estimate the simulation time from the values of PTD parameters.

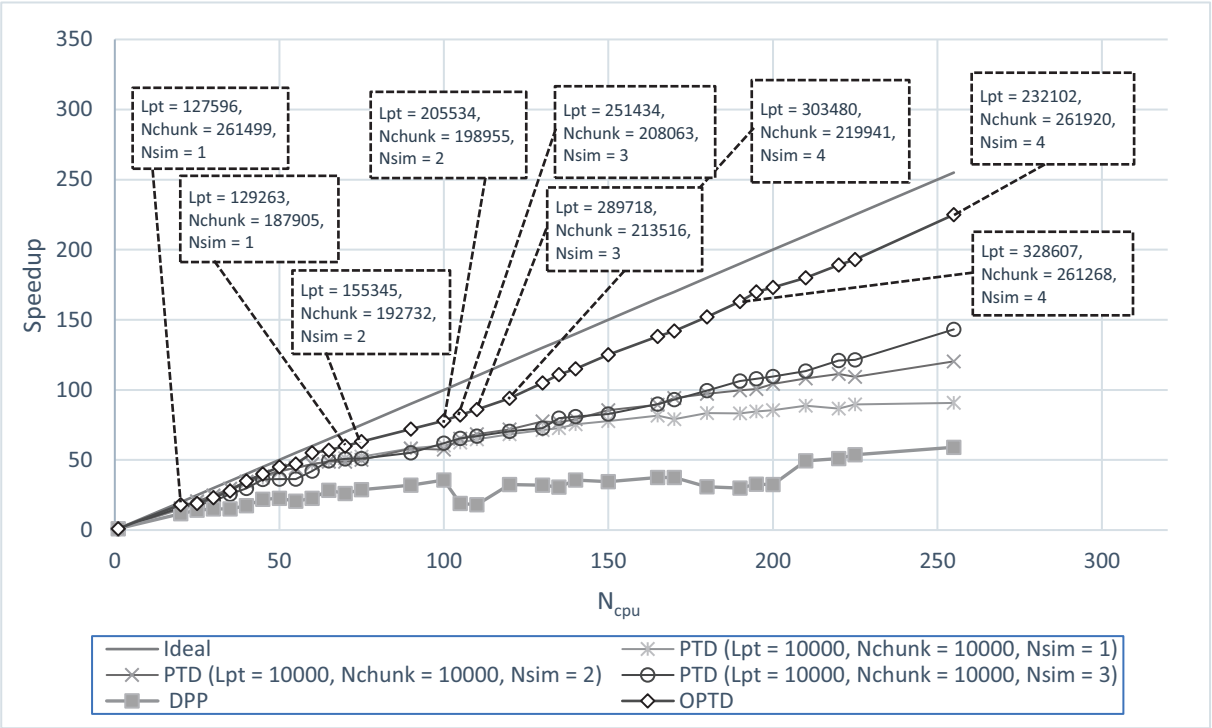


Figure 15. The speedup of OPTD, PTD and DPP. Some combinations of optimal parameters are also shown. The complete data set is available at https://github.com/srdjan034/optd.

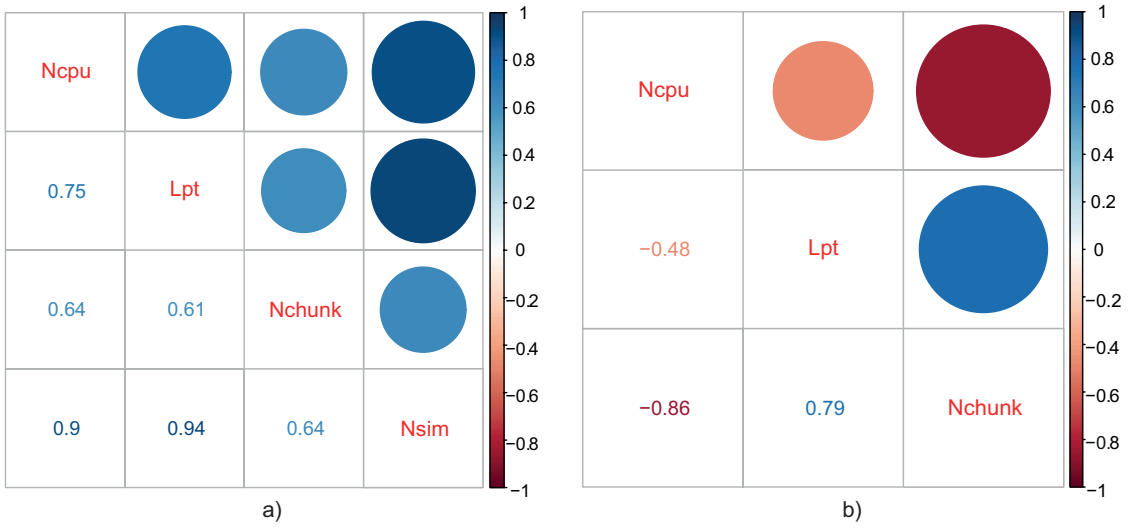


Figure 16. Pearson correlation of the optimal PTD parameters: *(a)* The whole data set, *(b)* Only records where $N_{sim} = 1$.

The results showed that the algorithm posses a good scalability potential showing speedups close to ideal. Our major innovation is reflected in creating a novel approach for parallelizing Lagrangian tracking in the form of an extensible framework. It should be simple to modify the model of particle behavior, as well as geometry. Last but not least, the presented approach could also be implemented with a variety of distributed programming technologies, not only MPI.

The main disadvantage is that the automatic training of the surrogate model, which takes a major part in the evaluation of the PTD parameter combinations, is still not supported. Also, we did not take into account the impact of the problem geometry on the performance. If we could calculate the probability of hitting a boundary, that value could represent the influence of the geometry within the surrogate model. However, computing such probability is not a trivial task and will be a subject of future research. Another direction of the future research will focus on the implementation of the presented method on the GPU platform, while the other could be an implementation of the algorithm in the HPC-in-the-Cloud environment with the capability of adding new workers at any time. In that case, the system has to adapt

dynamically to the new number of CPUs, determine their role and modify parameters to achieve optimal performance.

Acknowledgements

Part of this research is supported by The Ministry of Science in Serbia, Grants III41007, TR32036, and OI171021.

References

Beaudoin A, De Dreuzy JR and Erhel J (2007) An efficient parallel particle tracker for advection-diffusion simulations in heterogeneous porous media. In: European Conference on Parallel Processing. Springer, pp. 717–726.

Breuer M, Baytekin H and Matida E (2006) Prediction of aerosol deposition in 90 bends using les and an efficient lagrangian tracking method. Journal of Aerosol Science 37(11): 1407–1428.

Charles W, Van den Berg E, Lin HX, Heemink AW and Verlaan M (2008) Parallel and distributed simulation of sediment dynamics in shallow water using particle decomposition approach. Journal of Parallel and Distributed Computing 68(6): 717–728.

Hadka D (2016) High performance computing with the moea framework and ignite. CreateSpace Independent Publishing Platform .

Hall M, Frank E, Holmes G, Pfahringer B, Reutemann P and Witten IH (2009) The weka data mining software: an update. ACM SIGKDD explorations newsletter 11(1): 10–18.

Jin Y (2005) A comprehensive survey of fitness approximation in evolutionary computation. Soft computing 9(1): 3–12.

Larson DJ and Nasstrom JS (2002) Shared-and distributed-memory parallelization of a lagrangian atmospheric dispersion model. Atmospheric Environment 36(9): 1559–1564.

Lenôtre L (2016) A strategy for parallel implementations of stochastic lagrangian simulation. In: Monte Carlo and Quasi-Monte Carlo Methods. Springer, pp. 507–520.

Nikezić D and Stevanović N (2005) Radon progeny behavior in diffusion chamber. Nuclear Instruments and Methods in Physics Research Section B: Beam Interactions with Materials and Atoms 239(4): 399–406.

Nikezic D and Stevanovic N (2007) Behavior of 220rn progeny in diffusion chamber. Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment 570(1): 182–186.

Roberti DR, Souto RP, de Velho HFC, Degrazia GA and Anfossi D (2005) Parallel implementation of a lagrangian stochastic model for pollutant dispersion. International Journal of Parallel Programming 33(5): 485–498.

Salzberg SL (1994) C4. 5: Programs for machine learning by j. ross quinlan. morgan kaufmann publishers, inc., 1993. Machine Learning 16(3): 235–240.