electronics

MDPI

Article

Parallel Algorithm on Multicore Processor and Graphics Processing Unit for the Optimization of Electric Vehicle Recharge Scheduling

Vincent Roberge *, Katerina Brooks and Mohammed Tarbouchi

Department of Electrical and Computer Engineering, Royal Military College of Canada, Kingston, ON K7K 7B4, Canada; katerina.brooks@rmc-cmr.ca (K.B.); tarbouchi-m@rmc.ca (M.T.)
* Correspondence: vincent.roberge@rmc.ca; Tel.: +1-613-541-6000 (ext. 6492)

Abstract: Electric vehicles (EVs) are becoming more and more popular as they provide significant environmental benefits compared to fossil-fuel vehicles. However, they represent substantial loads on the power grid, and the scheduling of EV charging can be a challenge, especially in large parking lots. This paper presents a metaheuristic-based approach parallelized on multicore processors (CPU) and graphics processing units (GPU) to optimize the scheduling of EV charging in a single smart parking lot. The proposed method uses a particle swarm optimization algorithm that takes as input the arrival time, the departure time, and the power demand of the vehicles and produces an optimized charging schedule for all vehicles in the parking lot, which minimizes the overall charging cost while respecting the chargers' capacity and the parking lot feeder capacity. The algorithm exploits task-level parallelism for the multicore CPU implementation and data-level parallelism for the GPU implementation. The proposed algorithm is tested in simulation on parking lots containing 20 to 500 EVs. The parallel implementation on CPUs provides a speedup of 7.1x, while the implementation on a GPU provides a speedup of up to 247.6x. The parallel implementation on a GPU is able to optimize the charging schedule for a 20-EV parking lot in 0.87 s and a 500-EV lot in just under 30 s. These runtimes allow for real-time computation when a vehicle arrives at the parking lot or when the electricity cost profile changes.

Keywords: electric vehicles; graphics processing units; metaheuristic; multicore processors; parallel programming; parking lot; particle swarm optimization; recharge scheduling

check for updates

Citation: Roberge, V.; Brooks, K.; Tarbouchi, M. Parallel Algorithm on Multicore Processor and Graphics Processing Unit for the Optimization of Electric Vehicle Recharge Scheduling. *Electronics* **2024**, *13*, 1783. https://doi.org/10.3390/electronics13091783

Academic Editors: Nikolay Hinov, Darius Andriukaitis, Jožef Ritonja and Ping-Feng Pai

Received: 9 April 2024
Revised: 30 April 2024
Accepted: 2 May 2024
Published: 5 May 2024

Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

Electric vehicles (EVs) are becoming more and more popular due to their lower environmental footprint. However, they represent a significant load on the power grid, and this load is only going to increase in future years. As per the International Energy Agency, EVs consumed about 100 TWh in 2022, and this is expected to increase to over 380 TWh by the year 2032 [1]. To sustain this increased and dynamic demand, the power grid must be modernized, automated, and optimized. One way to contribute to this effort is to develop solutions for the EV charge scheduling problem. This problem focuses on generating an optimized charging schedule for all EVs within a system. The goal may be to reduce the peak power draw, maintain an acceptable voltage profile, or reduce the overall costs.

Like several scheduling problems, the problem of the EV charging schedule is non-convex, involves a large number of variables, and is NP-hard, which means that an optimal solution cannot be found in polynomial time [2]. In fact, for most realistically sized systems, it is not guaranteed to find the optimal solution. To be more precise, the problem of the EV charging schedule could be classified as a fully polynomial-time approximation scheme (FPTAS) [3], as it is somewhat similar to the well-known knapsack problem. This means that there exist polynomial time algorithms to find approximate solutions to the problem. However, these algorithms will only find approximative solutions. Based on [4], several algorithms have been tested to solve the EV charging schedule problem. These can be classified into three families: Exact optimization methods, heuristics, and metaheuristics. Exact methods include, among others, convex optimization methods [5], linear programming [6], and mixed integer nonlinear programming (MINLP) [7]. These are deterministic and very efficient when solving small instances of the problem. However, they can have difficulties coping with the complexity of realistic and large-scale problem sizes. Heuristics work by trial and error or by following a set of rules defined specifically to solve the problem at hand. An example of a heuristic-based approach is proposed in [8], where a graph search algorithm is used to find an optimized charging schedule. Two other examples of heuristics for the problem at hand are the first-in-first-serve (FIFS) method and the earliest-deadline-first (EDF) method [9]. Finally, metaheuristics are a class of optimization algorithms that can be applied to a wide range of optimization problems. These algorithms are generic, but a solution representation, a fitness function, and a search neighborhood must be defined specifically for the problem under consideration. Metaheuristics have the advantage of scaling very well to a large problem size. They can be used where classic and heuristic methods fail to perform adequately. Examples of metaheuristics for the EV charging schedule are particle swarm optimization (PSO) [10,11], and the genetic algorithm (GA) [12,13], although several others have been used. In fact, metaheuristics are probably the most commonly used methods for the EV charge scheduling problem. In their review paper, the authors of [4] cover the use of several metaheuristics for the problem of EV charging schedules and describe their advantages compared to other methods.

That being said, metaheuristics are no silver bullet. In fact, Wolpert and Macready, in their paper no free lunch theorems for optimization [14], establish through a set of theorems that performance improvement in one category of problems is typically balanced out by a decline in performance in another category. This means that metaheuristics are not better than other algorithms for all problems, but only for specific problems for which exact methods or heuristics do not provide adequate results. It also means that to apply the metaheuristic to a problem, one must define an efficient solution encoding, search neighborhood and fitness function. Another paper from Loscos et al. [15] demonstrates through a formal proof that the correlation between the inputs and the generated outputs remains undecidable in the context of local search heuristics and, by extension, metaheuristics. This means that metaheuristics are not guaranteed to generate optimal or even acceptable solutions in any given run due to the non-deterministic aspect of the algorithm. However, their advantage still remains invaluable in situations where other algorithms provide subpar results.

Because metaheuristics work by improving candidate solutions over a large number of iterations, they can also suffer from a long execution time, depending on the configuration parameters used. To avoid this problem, parallel implementations of metaheuristics have been proposed, which benefit from today's parallel processors. Examples are a parallel GA [16] and a parallel PSO [17], both implemented on multicore central processing units (CPUs). These parallel implementations use task-level parallelization where each thread performs a big chunk of work in parallel. Task-level parallelization is optimal for multicore CPUs where the number of cores is somewhat limited. Although the acceleration or speedup gained by the parallelization of multicore CPUs can be significant, there are parallelization implementations that can provide better speedup. One such technique is data-level parallelization on graphics processing units (GPUs). In data-level parallelization, each thread performs simple operations on a large quantity of data in parallel. Given that GPUs have a much higher number of cores than CPUs, data-level parallelization on GPUs has the potential to offer greater speedups. Examples of data-level parallelization on GPUs of the PSO are provided in [18] and of the GA, in [19]. These implementations are for other engineering problems, and therefore the solution encoding and the evaluation of the fitness function are different than in this paper. The parallelization strategies related to those aspects will therefore differ.

Given the advantage of metaheuristics compared to classic optimization methods and heuristics, this paper presents a PSO-based optimization algorithm for the problem of EV charge scheduling. The algorithm proposed is a multi-run PSO, which addresses the non-deterministic aspect of metaheuristics and provides a higher-quality result than a single-run PSO. Compared to other metaheuristics, the PSO is a natural fit for continuous optimization functions as particles evolve in a continuous multidimensional space [20]. To address the long execution time of the metaheuristic, the algorithm is parallelized on CPUs using task-level parallelization and on GPUs using data-level parallelization, and the performance results of both methods are thoroughly compared. The proposed multi-run PSO is tested in simulation using nine different scenarios containing 20 to 500 EVs. The results show that the proposed algorithm performs better than previously published algorithms in the literature, and scales well to large scenarios, which are not often used in the literature. The parallel implementation on CPUs leads to a 7.1x speedup, and the parallelization on a GPU leads to a 247.6x speedup, compared to a sequential execution on a CPU.

This paper makes three main contributions. Firstly, it describes from start to finish a solution for EV recharge scheduling optimization based on a multi-run PSO. Our proposed solution also includes a fitness function that encompasses the optimization objective and multiple constraints. Secondly, it proposes a method based on task-level parallelization to efficiently implement the multi-run PSO on multicore CPUs. This approach provides a significant speedup compared to sequential execution on a CPU, but the runtime remains too long to be used in real time. Thirdly, it proposes a method based on data-level parallelization to implement the multi-run PSO on a GPU. This method provides the greatest speedup and allows the algorithm to be used for real-time optimization.

The reminder of this paper is organized as follows. Section 2 presents previous works published on the topic of EV recharge scheduling optimization in order to appreciate the contribution of our paper. Section 3 discusses the materials and methods, which include the problem definition, a description of GPUs, the multi-run PSO, the fitness function, the parallelization of multicore CPUs, and the parallelization of a GPU. Section 4 presents the results based on five tests performed in simulation and finally, Section 5 concludes with a discussion on the relevance and significance of the proposed algorithm.

2. Previous Works

In this section, we present several works that have been published on the subject of EV recharge scheduling in order to better situate this paper amongst existing literature.

In [21], Chen et al. published a charge-scheduling algorithm for EVs in a parking lot. Their algorithm uses a simulated annealing (SA) metaheuristic and considers renewable energy sources such as photovoltaic panels and storage batteries. The SA is an algorithm that is inspired by the annealing process in metallurgy, and it usually improves a single candidate solution over several iterations. To make the algorithm more efficient, the authors use multiple candidate solutions and introduce the concept of elitism, where a good solution is allowed to continue to the next iteration even though it could be discarded as per the SA rules. They tested their algorithm in simulation on a 100-EV parking lot, and their multi-solution SA with elitism was revealed to be more efficient than a traditional SA.

To obtain a true population-based metaheuristic, one can refer to [13], in which Abdullah-Al-Nahid and Aziz proposed a GA for the EV recharge scheduling problem. The GA is a very popular metaheuristic inspired by the evolution of species. It improves a population of candidate solutions using operators such as crossover and mutation over several iterations. They tested their algorithm in simulation on a 30-EV parking lot and achieved positive results.

In [11], Zheng et al. present a PSO-based algorithm for the charging and discharging of EVs. They used a Monte Carlo method to estimate the charging load of EVs throughout the day, and they used a PSO to compute the optimal charging and discharging schedule of the EVs based on the time-of-use electricity pricing. PSO is another popular metaheuristic that has been used with great success in a wide range of engineering problems. The objective of this work is to reduce the peak-to-valley load difference for the grid and lower the charging cost for the customer, resulting in a multi-objective collaborative optimization.

To get the best of both worlds and benefit from the advantages of the GA and the PSO, the authors of [22] propose a hybrid GA-PSO algorithm for the problem of EV charging. Their problem description includes battery swapping stations where car batteries can be swapped for very fast recharging. They tested their algorithm on the IEEE-33 node distribution network with 100 groups of 10 EVs.

Another common metaheuristic used for the EV charge scheduling problem is the gravitational search algorithm (GSA), as it is proposed in [23]. This algorithm is similar to the PSO in the sense that particles (or candidate solutions) move in a multidimensional space and are attracted to each other. However, contrary to the PSO, where a particle is attracted only to its best previous position and the swarm's best particle, in the GSA, each particle is attracted to all other particles, and the attraction force is proportional to the quality of the particles and their distance. In [23], the author investigates three different decision functions for the GSA: The tangent, the sigmoid, and the round function, in order to identify which one of the three is best suited for the problem at hand. They test their GSA-based algorithm on the 34-node distribution network with fewer than 30 EVs.

Continuing in the frame of developing a more performant and faster converging algorithm, the author of [24] published a hybrid PSO-GSA-based method for the problem of EV charge scheduling. They conduct exhaustive testing using various benchmark functions to demonstrate the superiority of the algorithm compared to the two original algorithms before using it in simulation to optimize the charging schedule. Their optimization objective includes the overall charging cost and the load variance of the grid. They show that they were able to improve the operational stability of the grid and the economic benefit for the users.

In [25], Rho et al. focus on the problems of EV load estimation and EV charge scheduling, but rely on machine learning (ML) techniques instead of metaheuristics. In the first step, they use the XGBoost algorithm [26], which is a tree-based ML technique that uses a boosting method that learns residuals to estimate a function with great accuracy. In a second step, they use a convolutional neural network long short-term memory network (CNN-LSTM) to compute a charging/discharging schedule.

Two more works of interest on the topic of EV charge scheduling are [9,27], both published by Wu et al. These papers are interesting because they include the complete details of an example electricity cost profile and a single parking lot profile in the form of tables. This allows researchers to reuse their scenario and compare new methods to theirs. This is what we are doing in this paper, we are reusing the scenario from [9] to test our proposed parallel algorithm on multicore CPUs and a GPU for the optimization of EV recharge scheduling.

Despite the fact that the vast majority of papers on EV recharge scheduling use metaheuristics, and that metaheuristics are time consuming due to the vast majority of them being population based and typically requiring a large number of iterations, to our knowledge there has been little to no work conducted on developing parallel algorithms for the fast computation of EV recharge scheduling. However, there exist several parallel implementations of metaheuristics applied to other fields. In [28], a parallel PSO for a multicore CPU is proposed. The PSO population is divided into multiple sub-populations, and each thread runs an independent PSO algorithm on its own sub-population. At given intervals, the best particles from the sub-population are shared between the threads to allow cooperation. A similar approach was proposed in [29] to develop a parallel PSO and a parallel GA applied to the problem of trajectory planning for uncrewed air vehicles (UAV). In [30], the same strategy was used to parallelize a hybrid GA-PSO algorithm, still for the problem of UAV path planning. To achieve a higher performance than what is possible on a multicore CPU, one can turn to GPUs. A parallel GA on GPU for vehicle routing is proposed in [31], where a speedup of 1700x is achieved when compared to a sequential implementation on CPU. Other implementations of the GA on GPU are published in [32–34]. In all instances, data-level parallelism must be exploited in the algorithm to allow the implementation to benefit from the massively parallel architecture of the GPU. This represents a significant programming effort, but the speedup achieved is quite rewarding.

In this paper, we propose two parallel implementations of the PSO for the problem of EV recharge scheduling. The first implementation is for multicore CPUs, while the second is for GPUs. These parallel implementations allow a significant speedup of the algorithm, which is essential for large parking lots if one wants to run the algorithm in real time.

3. Materials and Methods

3.1. Problem Formulation

This section formulates the problem that this paper addresses. However, before the problem formulation is given, two terms must be defined. Firstly, an electricity cost profile is a structure that contains the cost of the electricity in kWh for each timeslot of the day. This is useful to represent variable electricity pricings. Secondly, a parking lot profile includes the arrival time, departure time and power demand (i.e., the amount of energy required to recharge the battery) for each EV that travels to or from the parking lot. In this work, it is assumed that there are enough charging stations in the parking lot to serve all the EVs. Given an electricity cost profile and a parking lot profile, the problem of EV charge scheduling is defined as follows:
minimize

$$C = \sum_{i=1}^{numEV} \sum_{t=arr_i}^{dep_i} x_{d_i}^t * ElecPrice_t \tag{1}$$

subject to

$$\sum_{t=arr_i}^{dep_i} x_{d_i}^t = dem_i \quad \forall i \in EV \tag{2}$$

$$x_{d_i}^t \leq limChr \quad \forall i \in EV, \forall t \in T \tag{3}$$

$$\sum_{i=1}^{numEV} x_{d_i}^t \leq limTf \quad \forall t \in T \tag{4}$$

where C is the overall cost for charging all EVs in the parking lot, $numEV$ is the number of EVs, arr_i and dep_i are the arrival and departure time of EV_i, $x_{d_i}^t$ is the electric quantity assigned to EV_i at timeslot t, $ElecPrice_t$ is the price of electricity in timeslot t, dem_i is the energy demand for vehicle i, $limChr$ is the charging rate limit for each vehicle, and $limTf$ is the transformer capacity limit for the entire parking lot. Equations (2)–(4) are the constraints to the problem.

In a word, the problem of EV charge scheduling consists of calculating a charging schedule for all EVs in the parking lot in order to minimize the overall cost of charging the vehicles while ensuring that each EV gets the required energy and respecting the maximum charging rate for each vehicle and the transformer capacity limit for the entire parking lot. This problem is large scale, nonlinear, non-convex, and NP-hard [2], hence the need for metaheuristic optimization, which is efficiently parallelized on parallel processors such as multicore CPUs and GPUs.

3.2. Graphics Processing Units

GPUs are coprocessors that were originally used specifically for graphics applications. They were initially built as parallel hardware pipelines to process a large throughput of graphics data efficiently. However, over the years, the hardware pipelines were replaced with a large number of programmable processors. The general architecture of an NVIDIA GPU is illustrated in Figure 1. The large number of compute cores is divided into streaming multi-processors (SM). Each core has a set of registers, and the cores within the same SM share a memory called shared memory. This small, but high-speed, memory is used to share data between threads within the same SM. The GPU has a level 2 cache that interfaces with the global memory. The global memory is shared by all the cores on the GPU. In this research, we use the NVIDIA RTX 4070 Ti (manufactured by NVIDIA Corp., sourced in Kingston, Canada) to run our experimental tests. This GPU has 7680 cores divided into 80 SMs, 48 MB of L2 cache, and 12 GB of global memory.

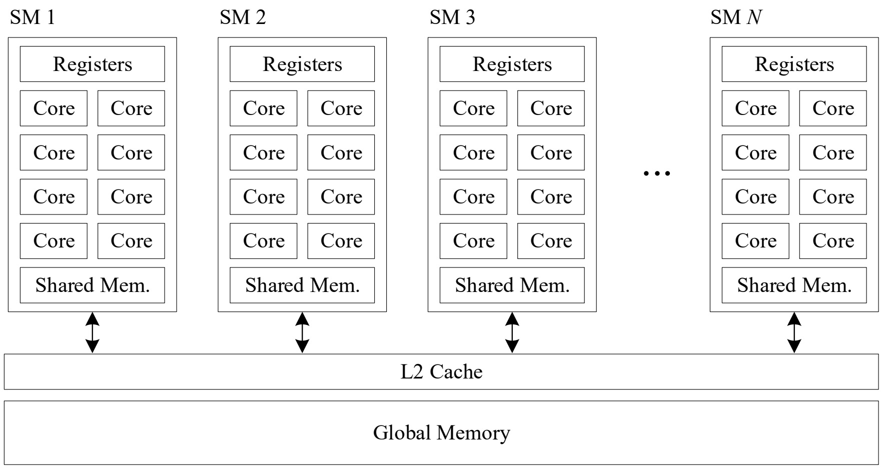


Figure 1. Generic architecture of an NVIDIA GPU.

When programming an NVIDIA GPU, one can use the CUDA compute framework [35]. The computer, or CPU, is referred to as the host, and the GPU is the device. A CUDA program consists of host code that calls CUDA functions (or kernels) to run code in parallel on the device. An illustration of the execution model of a CUDA program is shown in Figure 2. Here, the host code calls kernel 1, which launches a grid of threads on the device. The threads are organized in blocks. A grid can have up to three dimensions, and blocks can also have up to three dimensions. In Figure 2, both the grid and the blocks are two-dimensional. When launching a grid of threads, the blocks are mapped to the SM on the device. This means that threads from the same block can share data through shared memory within a kernel and can also be synchronized. However, threads from different thread blocks cannot be synchronized inside a kernel. They are synchronized at the end of the kernel only. To share data between threads from different blocks, one must use the global memory and use multiple kernels.

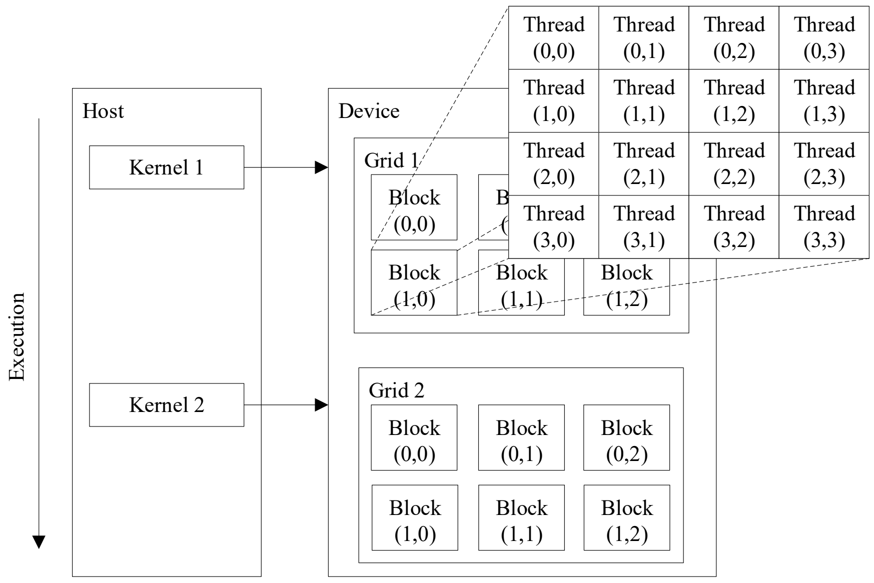


Figure 2. Execution model of the GPU.

3.3. Particle Swarm Optimization

The PSO is a population-based metaheuristic that was first published by Kennedy and Eberhard in 1995 [36]. The algorithm is inspired by the movement of a flock of birds or a school of fish. It simulates the movement of particles in a multidimensional search space. The position of the particles represents candidate solutions, and they evolve through an iterative process until optimized solutions are found. At each iteration, particles are attracted by their previous best position (local influence) and the swarm's best position (global influence).

In this work, we used an improved version of the PSO, which we call the multi-run PSO. Because the PSO is a non-deterministic algorithm, it usually produces different results each time the algorithm is run, and some instances return better results than others. For this reason, we wrap the original PSO in an outer loop that runs the PSO multiple times, and the best solution of all the runs is the solution returned by our multi-run PSO.

The flowchart of our multi-run PSO is shown in Figure 3. Each run starts at line 2 by randomly initializing the position $\boldsymbol{x_t}$ and velocity $\boldsymbol{v_t}$ of the particles. The cost of each particle is then evaluated at line 3. Based on the cost calculated, the particles' best previous positions $\boldsymbol{b}_t$ are updated at line 4 and the swarm's best position $\boldsymbol{g}_t$ is updated at line 5. The velocity and position of the particles are then updated at lines 6 and 7 using:

$$\boldsymbol{v}_{t+1} = \omega \boldsymbol{v}_t + c_1 \boldsymbol{r}_1 (\boldsymbol{b}_t - \boldsymbol{x}_t) + c_2 \boldsymbol{r}_2 (\boldsymbol{g}_t - \boldsymbol{x}_t) \tag{5}$$

$$\boldsymbol{x}_{t+1} = \boldsymbol{x}_t + \boldsymbol{v}_{t+1} \tag{6}$$

where all bold variables are vectors, $\boldsymbol{v}$ is the velocity of the particle; $\boldsymbol{x}$ is its position; $\boldsymbol{b}$ is the best position previously occupied by the particle; $\boldsymbol{g}$ is the best position previously occupied by any particle of the swarm; $\boldsymbol{r}_1$ and $\boldsymbol{r}_2$ are vectors of random values between 0 and 1; and ω, c_1 and c_2 are the inertia, the personal influence, and the social influence parameters, respectively. Finally, t is the time or iteration index. The termination criterion is verified at line 8. In our implementation, we have used a fixed number of iterations that was selected experimentally so that good solutions are produced for all the problem sizes studied. Line 9 is where the best solution tracked by the multi-run PSO is updated. Line 10 is the termination criterion for the outer loop of the multi-run PSO. In our case, it is also based on a fixed number of iterations. This has the advantage of guaranteeing a known runtime for the algorithm for a given instance size and characteristics, as well as the computing environment. Finally, the best result is returned by the algorithm at line 11.

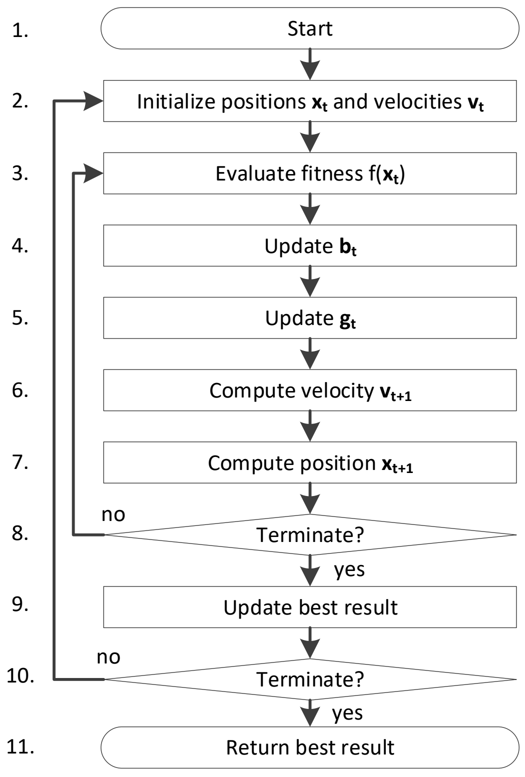

1.
Start
2.
Initialize positions $\mathbf{x_t}$ and velocities $\mathbf{v_t}$
3.
Evaluate fitness f($\mathbf{x_t}$)
4.
Update $\mathbf{b_t}$
5.
Update $\mathbf{g_t}$
6.
Compute velocity $\mathbf{v_{t+1}}$
7.
Compute position $\mathbf{x_{t+1}}$
8.
no
Terminate?
yes
9.
Update best result
10.
no
Terminate?
yes
11.
Return best result

Figure 3. Flowchart of the multi-run PSO.

3.4. Solution Encoding

The input to the PSO is provided in the form of an electricity cost profile and a parking lot profile. As explained earlier, an electricity cost profile contains the cost of the electricity in kWh for each timeslot of the day. A parking lot profile includes for each car their arrival time, their departure time, and their power demand (i.e., the amount of energy required to recharge the battery). Candidate solutions or particles used by the multi-run PSO are encoded as:

$$\boldsymbol{X} = \left\{ x_1^{arr_1}, \ldots, x_1^{dep_1}, x_2^{arr_2}, \ldots, x_2^{dep_2}, \ldots, x_i^t, \ldots, x_{numEV}^{arr_{numEV}}, \ldots, x_{numEV}^{dep_{numEV}} \right\} \tag{7}$$

where x_i^t is the electric quantity assigned to EV_i at timeslot t. There is an x for each timeslot for which the vehicle is parked in the parking lot. The value of x varies between 0 and 1 and represents the proportion of the charging demand of the vehicle. To decode a candidate solution, one must use the following equation:

$$x_{d_i}^{\ t} = \left(\frac{x_i^t}{\sum_{t=arr_i}^{dep_i} x_i^t} \right) dem_i \tag{8}$$

where $x_{d_i}^{\ t}$ is the decoded version of x_i^t and dem_i is the charging demand for vehicle i. In the event that all x_i^t are equal to zero, the decoding resets these values to 0.5 to avoid a division by zero. This decoding ensures that the candidate solution always fulfill the power demand of the vehicle equality defined in Equation (2) holds true for all vehicles i.

3.5. Fitness Function

The fitness function is what guides the multi-run PSO during the optimization process. It is composed of an optimization objective and constraints. The optimization objective consists of minimizing the cost of the electricity purchased by the owners of the EVs, as defined in Equation (1). There are three constraints to the problem. The one defined in Equation (2) is satisfied by the solution encoding. The next two are integrated into the fitness function. The constraint at Equation (3) specifies that an EV cannot request more power than what is available at a charging station or what the charging port can provide during any single timeslot. The constraint at Equation (4) specifies that all the EVs in the parking lot cannot request more power than the transformer capacity during any single timeslot. These constraints are integrated into the fitness function by using penalty terms. The penalty term associated with the constraint at Equation (3) is defined as:

$$d_{limChr} = \sum_{i=1}^{numEV} \sum_{t=arr_i}^{dep_i} d_i^t \tag{9}$$

where

$$d_i^t = \begin{cases} 0, & x_{d_i}^{\ t} \le limChr \\ \left(x_{d_i}^{\ t} - limChr \right)^2, & x_{d_i}^{\ t} > limChr \end{cases} \tag{10}$$

For every timeslot in which an EV is present, this checks if the charging power requested by an EV is higher than the available power at the charger. If it is, the square of the difference is added to a sum, which forms the penalty term d_{limChr}. The penalty term associated with the constraint at Equation (4) is defined as:

$$d_{limTf} = \sum_{t=1}^{T} e_t \tag{11}$$

where

$$e_t = \begin{cases} 0, & \sum_{i=1}^{numEV} x_{d_i}^{\;t} \le limTf \\ \left(\sum_{i=1}^{numEV} x_{d_i}^{\;t} - limTf\right)^2, & \sum_{i=1}^{numEV} x_{d_i}^{\;t} > limTf \end{cases} \tag{12}$$

For every timeslot, this checks if the charging power requested by all vehicles present in the parking lot is greater than the transformer limit. If it is greater, the square of the difference is added to a sum, which forms the penalty term d_{limTf}.

A combined penalty term ρ is then computed by summing the two terms:

$$\rho = d_{limChr} + d_{limTf} \tag{13}$$

Now that the penalty term has been calculated, the fitness function $F(\boldsymbol{X})$ can be computed by integrating both the objective function and the penalty term, as follows:

$$F(\boldsymbol{X}) = \begin{cases} 0 + \frac{1}{1+\rho}, & \rho \ge 0 \\ 1 + \frac{1}{1+C}, & \rho = 0 \end{cases} \tag{14}$$

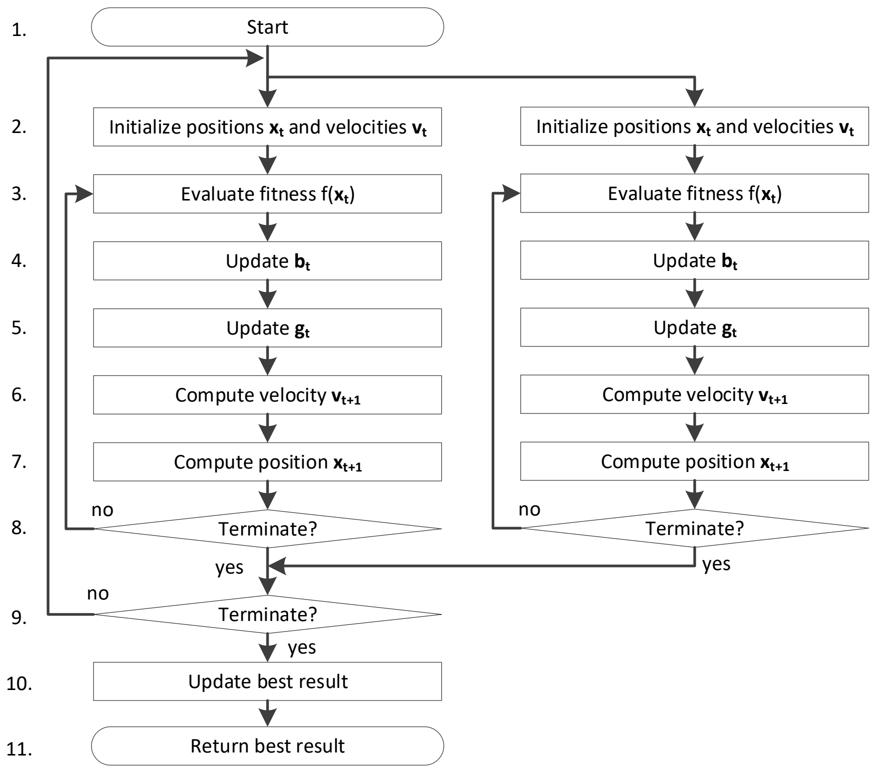
When there is a penalty value greater than 0, the penalty term is considered in the function, and the fitness has a value between 0 and 1. This represents infeasible solutions, as one or more of the two constraints were violated. Despite being infeasible, solutions with a lower level of violation result in a better fitness. This is important to guide the multi-run PSO towards feasible solutions at the initial stage of the optimization process. When there is no penalty, the cost term is considered, and fitness has a value between 1 and 2. This represents a feasible solution. Moreover, fitness increases inversely with cost, meaning that solutions with lower costs have higher fitness.

3.6. Parallelization on Multicore CPU Using OpenMP

The first parallelization technique employed in this paper is parallelization on multicore CPUs using OpenMP. OpenMP [37] is a popular specification for parallel programming. It allows the easy creation and management of threads on multicore CPUs. One key feature of OpenMP is that it can run for-loops in parallel by running each iteration of the loop using a different thread. This, of course, is feasible only when the iterations do not depend on each other. In our multi-run PSO, the outer loop fulfills this requirement as each run of the PSO is completely independent. For this reason, we have parallelized our multi-run PSO by running the iterations of the outer for-loop in parallel. The flowchart of the resulting parallel implementation is shown in Figure 4 for the case of two threads. In the flowchart, we can note that the outer loop still appears, but iterations are completed by multiple independent threads. Before the start of the outer loop, an array of the same dimension as the number of outer loop iterations is created in memory to store the best fitness and position for each iteration. This ensures that each iteration of the outer for-loop does not share any data and is fully independent. At the end of the parallel for-loop, the main thread iterates through this array to find the very best fitness and position, which is returned by the algorithm. Because the outer loop has a large number of iterations, a large number of threads can be used in the parallel implementation.

3.7. Parallelization on GPU Using CUDA

The parallelization on a multicore CPU is a task-level parallelization. Each thread performs a big chunk of work in parallel. In the case of the multi-run PSO, each thread performs one or more runs of the PSO. The performance gain will be significant but cannot be higher than the number of outer-loop iterations. If higher performance is desired, a data-level parallelization is required where each step of the original algorithm is parallelized so that it executes in parallel on the data. Because there are a large number of particles and each particle has a large dimension, the performance gain obtained by the data-level parallelization is expected to be much higher than that of the task-level parallelization.



1. Start
2. Initialize positions $\mathbf{x}_t$ and velocities $\mathbf{v}_t$
3. Evaluate fitness f($\mathbf{x}_t$)
4. Update $\mathbf{b}_t$
5. Update $\mathbf{g}_t$
6. Compute velocity $\mathbf{v}_{t+1}$
7. Compute position $\mathbf{x}_{t+1}$
8. Terminate?
no
yes
9. Terminate?
10. Update best result
11. Return best result

Figure 4. Flowchart of the parallel multi-run PSO using OpenMP.

In our second parallel design, we use the CUDA compute framework to develop a data-level parallel implementation of the multi-run PSO. Instead of parallelizing the outer-loop iteration, we parallelize each step of the original PSO on the GPU and leave the outer-loop sequential.

Before we can discuss in detail the implementation of each step, we must present two parallel primitives that are used extensively in the proposed CUDA implementation. The first parallel primitive is the parallel map illustrated in Figure 5. In this example, it performs the square of the values in the input vector using one thread per value. The parallel map primitive can be used for any operation that maps N input values to N output values using one thread per value. Compared to a sequential loop, which takes N steps to process the input vector, the parallel map primitive takes a single step, provided that there are enough compute cores on the device to run the operation. Because of this, the parallel map primitive is extremely efficient and typically leads to substantial speedups in a parallel implementation.

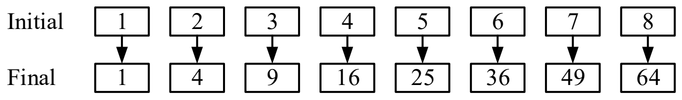


Figure 5. Parallel map primitive squaring each value of the input vector.

The second parallel primitive illustrated in Figure 6 is the parallel reduction. It maps N input value to a single output. In this example it is used to find the maximum value, but it could also be used to compute the sum or the product of an input vector. Compared to a loop which would take N steps, the parallel reduction primitive takes $log_2(N)$ steps. As an example, for an input vector of 1024 elements, only 10 steps would be required to perform the reduction, provided that there are enough compute cores on the device.

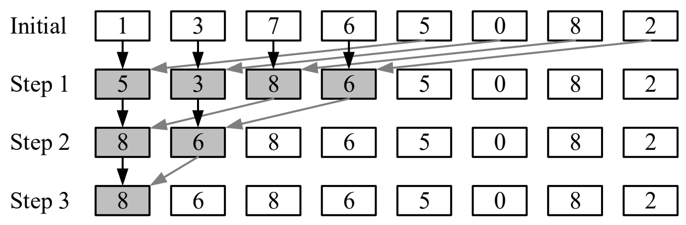


Figure 6. Parallel reduction primitive to find the maximum value.

Now that these two parallel primitives have been introduced, we can go over the details of the parallel implementation on a GPU, for which the flowchart is shown in Figure 7. In the GPU implementation, the iterations of the outer loop of the multi-run PSO are completed sequentially, but the steps within each PSO run are completed in parallel. The algorithm starts on the CPU, or host, which controls the sequence of the parallel kernels that are called on the GPU, or device. For maximum performance, all the steps of the algorithm are parallelized, and limited interaction between the CPU and the GPU is required during the optimization process.

In step 2, the initialization of the particles' position and velocity is parallelized using a parallel map primitive with one thread per dimension of each candidate solution. As an example, for 1000 solutions, each with a dimension of 85, a total of 85,000 threads are launched. Each thread initializes its individual random number state, which is kept in global memory, and uses this state to initialize the position and velocity of its associated particle.

Next comes the computation of the fitness for each candidate solution at step 3. In this case, we use one thread block per solution and one thread per vehicle. Each thread decodes the candidate solution for a vehicle and verifies if the charge rate has been exceeded at any timeslot. The kernel uses an atomic add to compute the transformer demand in each timeslot. Each thread also computes the electricity cost to recharge its associated vehicle. Three parallel reduction primitives are used to compute the charge rate excess term from Equation (10), the transformer limit excess term from Equation (12), and the overall cost for the parking lot. Finally, the fitness for the candidate solutions is computed by thread 0 of each block using the aggregated values from the parallel reductions.

Step 4 consists of the update of each particle's best previous position. This is parallelized using a parallel map primitive and one thread per dimension of the candidate solutions. Based on the thread identifier (ID), each thread verifies if the fitness of the newly calculated solution is greater than the fitness of its current previous best and updates one dimension of its current best.

Steps 5 and 6 consist of a 2-kernel reduction operation to find the index of the particle with the highest fitness. In the first kernel, a parallel reduction primitive is used in each block to find the maximum value within the block. In the second kernel, another parallel reduction primitive is used to find the maximum value from the output of the previous kernel. A 2-kernel implementation is required because blocks in CUDA cannot be synchronized or share data within a single kernel.

Now that the index of the best particle has been found, step 7 updates the swarm's best particle using a parallel map operation with one thread per dimension of the candidate solution.

The velocity and position of every particle are updated in step 8 using another parallel map primitive with one thread per dimension of each candidate solution.

The termination criterion, which consists of a fixed number of iterations, is verified at step 9. Once the PSO is complete and the termination criteria are met, the swarm's best particle is copied from the GPU's global memory to the CPU to be used by the outer loop of the multi-run PSO.

Throughout the implementation of the parallel PSO on the GPU, the number of threads used in each kernel in each step was always maximized. This ensures that the massively parallel architecture of the GPU is utilized to its maximum, and also provides scalability for future GPUs, which are likely to contain more cores.

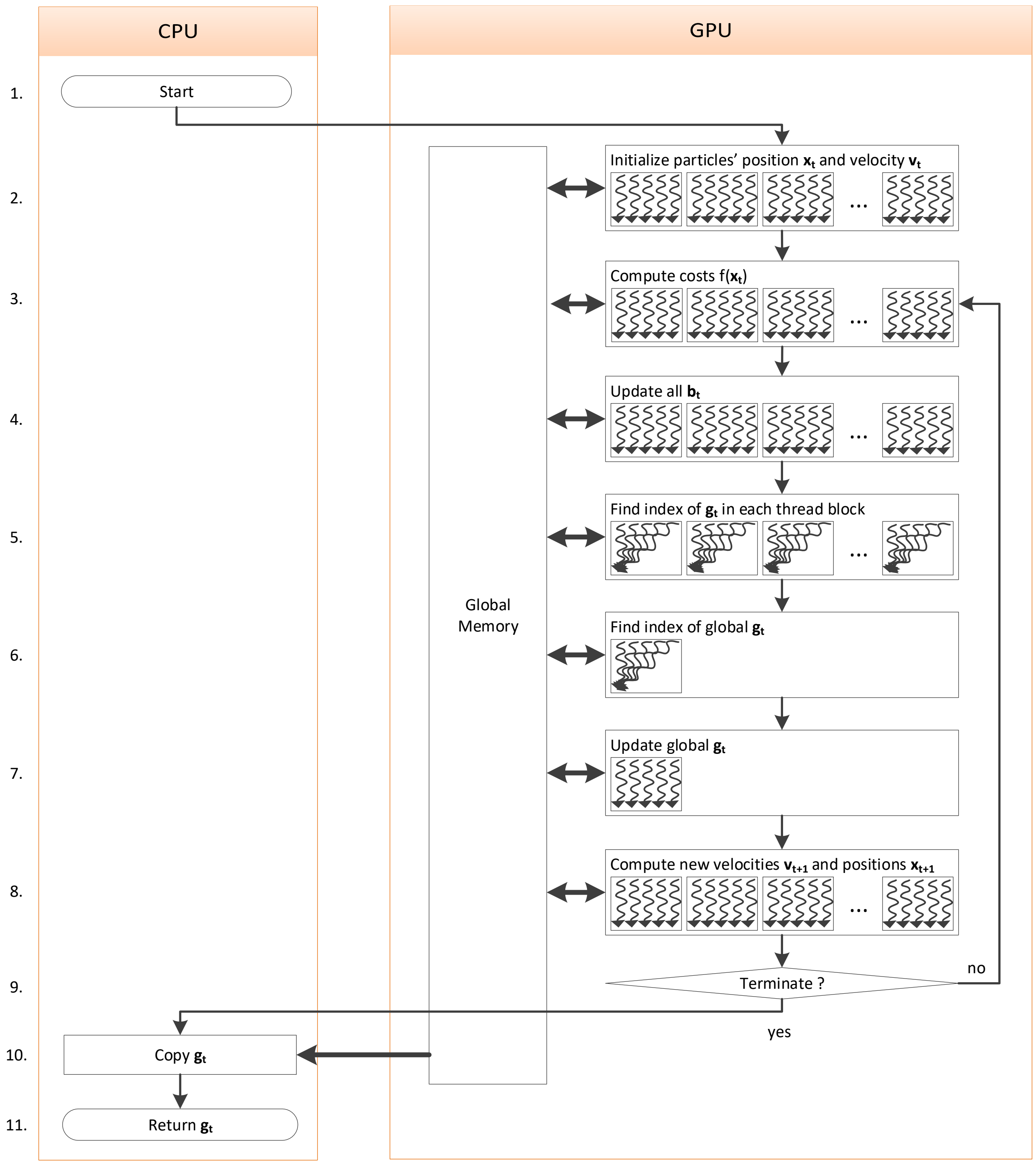

CPU
GPU
1.
2.
3.
4.
5.
6.
7.
8.
9.
10.
11.
Start
Initialize particles' position $\mathbf{x}_t$ and velocity $\mathbf{v}_t$
Compute costs f($\mathbf{x}_t$)
Update all $\mathbf{b}_t$
Find index of $\mathbf{g}_t$ in each thread block
Find index of global $\mathbf{g}_t$
Update global $\mathbf{g}_t$
Compute new velocities $\mathbf{v}_{t+1}$ and positions $\mathbf{x}_{t+1}$
Terminate ?
no
yes
Global Memory
Copy $\mathbf{g}_t$
Return $\mathbf{g}_t$

Figure 7. Flowchart of the parallel multi-run PSO on GPU using CUDA.

4. Results

The proposed parallel PSO for parking lot EV recharging schedule optimization is validated using nine tests. In the first test, we run the proposed algorithm for a 20-EV parking lot and compare the results to previously published results in [9] to demonstrate the expected functioning of the algorithm. In the second test, we run the algorithm for larger parking lots, ranging from 100 to 500 EVs, to demonstrate the scalability of the method. In the third test, we demonstrate the advantage of the multi-run PSO compared to the single-run PSO. In the fourth test, we measure the runtime and speedup of the first parallelization technique, which is the parallel PSO on multicore CPUs using OpenMP.

Finally, in the fifth test, we measure the runtime and speedup of the second parallelization technique, which is the parallel PSO on GPU using CUDA.

The experimental setup used to run the simulations consists of a Dell 7920 Workstation (manufactured by Dell Inc., sourced in Kingston, Canada) equipped with dual Intel Xeon Silver 4214R CPUs. Each CPU has 12 hyperthreaded cores, for a total of 24 hyperthreaded cores. The CPUs are clocked at 2.40 GHz. The workstation is also equipped with an NVIDIA RTX4070 Ti GPU (manufactured by NVIDIA Corp., sourced in Kingston, Canada) with 7680 CUDA cores clocked at 2.31 GHz and 12 GB of graphics GDDR6X RAM memory. The workstation has 128 GB of DDR4 RAM memory.

4.1. Test 1: Optimizing the Schedule for a 20-EV Parking Lot

The first simulation test consists of using the proposed parallel PSO to compute an optimized recharging schedule for a 20-EV parking lot. To allow comparison with results published in the literature, we have used the same electricity cost profile and EV parking profile as in [9]. The electricity cost profile is listed in Table 1. It shows the cost of the electricity in USD/kWh during the 10 time periods during which the parking lot is open. The reader can note that the cost is at its maximum at timeslot 3 and timeslot 8, times during which recharging of the EVs should be avoided as much as possible due to the higher costs. The next input to the simulation is the EV parking lot profile listed in Table 2. For each vehicle, the profile includes its arrival time, its departure time, and its power demand in kWh. This power demand represents the energy required to recharge the vehicle. The EV charging rate limit in a single time interval is 9.6 kW, and the base transformer limit value for the entire parking lot is 60 kW. The parallel PSO is configured using the parameters listed in Table 3. The inertia, personal, and social influence parameters have been set using values recommended from [38], which promote the convergence of the algorithm. For the number of particles and the number of inner- and outer-loop iterations, these values have been set experimentally to perform well in all test cases used in this study. The test is repeated 100 times to gather statistical data.

Table 1. Electricity cost (USD/kWh) (reproduced from [9]).

Time	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Electricity Cost	0.1	0.2	0.4	0.2	0.1	0.1	0.2	0.4	0.2	0.1

Table 2. EV parking lot profile (reproduced from [9]).

EV	**Arrival Time**	**Departure Time**	**Demand (kWh)**	**EV**	**Arrival Time**	**Departure Time**	**Demand (kWh)**
1	1	3	18	11	3	8	26
2	3	5	15	12	2	6	17
3	1	5	25	13	5	8	15
4	2	4	18	14	4	8	16
5	2	3	15	15	3	7	14
6	6	10	22	16	5	8	12
7	7	8	14	17	4	7	16
8	8	10	16	18	5	9	19
9	7	8	10	19	1	10	28
10	6	9	20	20	1	5	16

The results of one simulation are shown in Table 4. For each EV, this table lists the power demand in kW during each timeslot (or hour). The power demand for all EVs was satisfied. As an example, EV1 uses 9.60 kW during the first hour and 8.40 kW during the second hour, which sums to the desired 18 kWh demand as listed in Table 2. It is the same for all other EVs. We can note that minimal recharging is performed during hours 3 and 8 due to the higher electricity cost, which shows the proper, well-functioning optimization behavior of the algorithm. From the data listed in Table 4, it is also apparent that the constraints have been respected. The maximum power demand for each EV in each timeslot never exceeds the maximum of 9.6 kWh, and the overall demand for the entire parking lot never exceeds the base transformer limit of 60 kW. The cost of this recharging schedule is USD 53.72. It is lower than what has been published in [9] for various metaheuristic-based algorithms, which ranged from USD 53.77 to USD 59.45. The difference between our proposed GPU-based PSO and the best algorithm published in [9] is small, but the true advantage of our proposed GPU-based PSO is that it is able to scale to very large parking lots, which was not demonstrated in [9]. When compared to heuristic methods, the difference is more prominent in favor of our multi-run PSO. In [9], the authors reported a cost of USD 70.10 for a random search (RS), a cost of USD 73.52 for first-in-first-serve (FIFS), and a cost of USD 73.69 for earliest-deadline-first (EDF).

Table 3. Simulation parameters.

PSO Parameters	Value
PSO inertia (ω)	0.7298
PSO personal influence (c_1)	1.4960
PSO social influence (c_2)	1.4960
Number of particles	1000
Number of iterations (inner loop)	5000
Number of iterations (outer loop)	10

Table 4. Optimized schedule for 20 EVs as calculated by the proposed PSO algorithm.

EV	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	Total
EV1	9.60	8.40	0.00	-	-	-	-	-	-	-	18.00
EV2	-	-	0.00	7.64	7.36	-	-	-	-	-	15.00
EV3	9.60	3.88	0.00	5.70	5.83	-	-	-	-	-	25.00
EV4	-	8.45	0.00	9.55	-	-	-	-	-	-	18.00
EV5	-	9.60	5.40	-	-	-	-	-	-	-	15.00
EV6	-	-	-	-	-	4.58	2.36	0.00	5.46	9.60	22.00
EV7	-	-	-	-	-	-	9.60	4.40	-	-	14.00
EV8	-	-	-	-	-	-	-	0.00	6.40	9.60	16.00
EV9	-	-	-	-	-	-	9.60	0.40	-	-	10.00
EV10	-	-	-	-	-	7.24	7.10	0.00	5.65	-	20.00
EV11	-	-	0.00	3.83	8.01	7.73	6.42	0.00	-	-	26.00
EV12	-	1.72	0.00	3.60	5.12	6.56	-	-	-	-	17.00
EV13	-	-	-	-	7.47	7.53	0.00	0.00	-	-	15.00
EV14	-	-	-	1.38	4.04	5.79	4.79	0.00	-	-	16.00
EV15	-	-	0.00	4.31	3.00	4.13	2.56	-	-	-	14.00
EV16	-	-	-	-	4.86	2.97	4.17	0.00	-	-	12.00
EV17	-	-	-	5.73	4.02	4.73	1.52	-	-	-	16.00
EV18	-	-	-	-	5.05	5.61	5.61	0.00	2.74	-	19.00
EV19	9.60	1.52	0.00	0.00	1.26	3.14	2.87	0.00	0.01	9.60	28.00
EV20	9.60	1.90	0.00	0.52	3.97	-	-	-	-	-	16.00
Total	38.40	35.47	5.40	42.26	60.00	60.00	56.60	4.80	20.26	28.80	352.00

4.2. Test 2: Optimizing the Schedule for Larger Parking Lots

For the second test, we optimize larger parking lots containing 40 to 500 EVs. The parking lot profile for these larger parking lots is generated randomly (Supplementary Materials). The PSO is configured as per Table 3. The algorithm is run 100 times to obtain statistical results, which are listed in Table 5. For each scenario, we have listed the best, worst, median, and mean costs with the standard deviation. Although the 20-EV scenario has been discussed in the previous test, we show the results in Table 5 for the sake of completeness.

Table 5. Result of the parallel multi-run PSO for larger size parking lot.

Parking Lot Size (Number of EVs)	Dimension	Best (USD)	Worst (USD)	Median (USD)	Mean (USD)	Standard Deviation (USD)
20	85	53.72	53.72	53.72	53.72	0.00
40	171	120.29	120.71	120.30	120.36	0.10
60	270	174.62	176.91	175.80	175.76	0.58
80	356	251.23	256.01	253.69	253.67	1.00
100	394	300.97	306.87	304.45	304.35	1.30
200	898	692.19	716.31	706.03	705.67	4.64
300	1251	969.72	999.96	989.44	988.60	6.25
400	1700	1425.02	1459.22	1444.29	1443.87	7.21
500	2099	1821.19	1859.99	1846.07	1845.41	7.88

Regarding the performance of the algorithm, firstly, one can note that the dimension of the problem is very large. There is one dimension for each timeslot in which an EV is parked in the parking lot. As most EVs are parked for more than one timeslot, the dimension of the problem is much larger than the number of EVs. This high dimension speaks to the difficulty of the problem and explains why large numbers of particles and iterations are required to run the algorithm. Secondly, despite the large dimension, the proposed algorithm is very consistent at finding optimized solutions. This is clear in the 20-EV scenario, where the best and worst solutions have the same value. This means that the algorithm found a schedule that costs exactly USD 53.72 for all 100 runs. For the larger scenarios, there are differences between the best and worst solutions, but the standard deviation is always very small. The largest standard deviation is USD 7.88 for the 500-EV scenario, which represents 0.43% of the best solutions. This means that the proposed algorithm is able to find good-quality solutions in a very consistent manner.

4.3. Test 3: Comparison of the Single-Run and Multi-Run PSO

The third test consists of demonstrating the advantage of the multi-run PSO compared to a single-run PSO. Because the PSO is a non-deterministic algorithm, it can return suboptimal solutions in any single run. To address this limitation and guarantee higher-quality solutions, it is important to run the PSO multiple times and return the best solution found after all runs. In this test, we configure the single-run PSO to use 50,000 iterations, and the multi-run PSO is configured as in the previous tests with 10 rounds of 5000 iterations. This way, both configurations perform the same amount of work, but the multi-run benefits from restarting the search at the beginning of every run. The test is repeated 100 times for each scenario to obtain statistical results. The results are listed in Table 6, with the best mean and standard deviation highlighted in bold. One can note that for most of the scenarios, the multi-run PSO achieved a lower mean cost than the single-run PSO. Only for the largest scenarios is the single-run PSO better. Moreover, the standard deviation is smaller for the multi-run PSO in all instances. This demonstrates the advantage and the motivation for using a multi-run PSO.

4.4. Test 4: Performance of the Parallel PSO for Multicore CPU Using OpenMP

In the fourth test, we measure the runtime and speedup provided by the first parallel implementation of the proposed PSO-based algorithm, which consists of a multi-threaded implementation using OpenMP. In this implementation, the multiple CPU threads run the iterations of the outer loop of the algorithm concurrently. The minimum runtime and maximum speedup are expected when the number of threads approaches the number of iterations of the outer loop. Since there is no interaction between the iterations of the loop, a linear speedup is expected, where the speedup should be equal to the number of threads used. The system used for this test is the multicore workstation for which the specifications have been provided at the beginning of this section. The scenario used is the 20-EV parking lot scenario, and the number of threads is varied between 1 and 10. Since the parameters of the PSO remain the same as those listed in Table 3. Since the number of outer-loop iterations is set to 10 and these iterations are to be completed in parallel using one thread per iteration, the maximum speedup is expected using 10 threads.

Table 6. Comparison of the results for the single-run and multi-run PSO.

Parking Lot Size (Number of EVs)	Dimension	Single-Run PSO		Multi-Run PSO	
		Mean (USD)	Standard Deviation (USD)	Mean (USD)	Standard Deviation (USD)
20	85	54.10	0.44	53.72	0.00
40	171	121.42	0.79	120.36	0.10
60	270	176.49	1.16	175.76	0.58
80	356	253.80	1.38	253.67	1.00
100	394	306.30	3.70	304.35	1.30
200	898	708.01	6.86	705.67	4.64
300	1251	998.68	11.64	988.60	6.25
400	1700	1439.02	13.28	1443.87	7.21
500	2099	1841.24	11.13	1845.41	7.88

The runtimes and speedup are shown in Figure 8. These are the average values of 10 runs. The PSO-based optimization algorithm took 187.6 s to run the scenario with one thread and only 26.3 s to run the same scenario using 10 threads. This represents a speedup of 7.1x, which is close to the number of threads used, meaning that a linear or perfect speedup was almost achieved. This demonstrates the appropriateness of the parallelization technique used to parallelize the algorithm using multicore CPUs. However, despite the significant performance gain, the final runtime of 26.3 s still remains too high for a small parking lot. This means that when an EV arrives in the parking lot, it takes 26.3 s to recompute the optimized schedule. Moreover, although not shown here, this runtime increases significantly when the size of the parking lot increases. That being said, it is possible to further reduce the runtime of the proposed algorithm through a parallel implementation on a GPU.

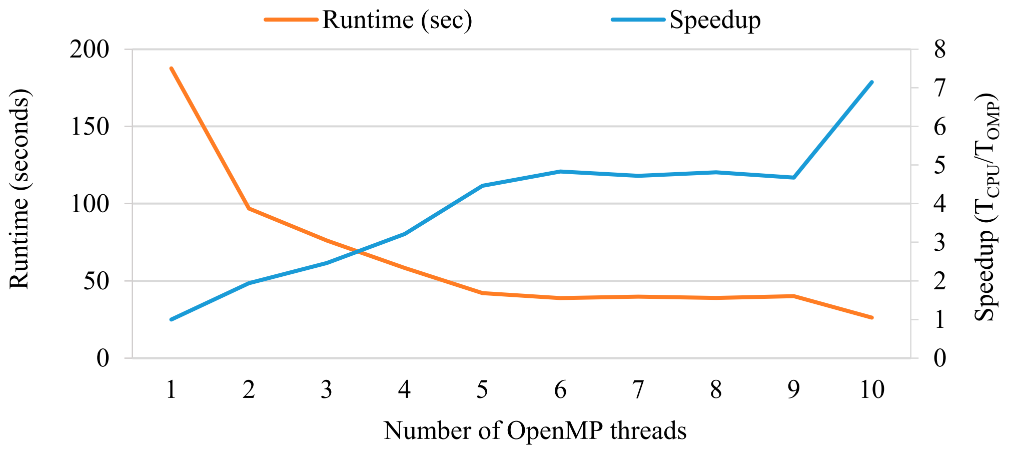


Figure 8. Runtime and speedup of the parallel implementation on multicore CPU using OpenMP.

4.5. Test 5: Performance of the Parallel PSO for GPU Using CUDA

The final test consists of measuring the runtime and speedup of the parallel implementation on a GPU. For this test, we use the 20-, 100-, 200-, and 500-EV scenarios and run the algorithm 10 times. The average runtimes and speedups are shown in Figure 9. The runtimes for the sequential version on a CPU are 187.6 s for the 20-EV scenario and 2506.1 s for the 500-EV scenario. In comparison, the runtimes for the GPU implementation are 0.87 s and 29.3 s, respectively. This results in speedups of 215.6x for the 20-EV scenario and 85.5x for the 500-EV scenario. The maximum speedup of 247.6x is achieved using the 100-EV scenario. This is due to the level of parallelism being sufficiently high to fully utilize the massively parallel architecture of the GPU. After 100 EVs, the amount of data processed by each CUDA block becomes too high, and the performance suffers, especially because of the atomic operation used in the evaluation of the fitness function. This explains why the speedup decreases after 100 EVs. It is now clear, especially for larger parking lots, that a sequential implementation on the CPU takes too long to be used in real time, whereas a parallel implementation on the GPU can be completed in real time. These measurements truly show the advantage of the parallel implementation on the GPU.

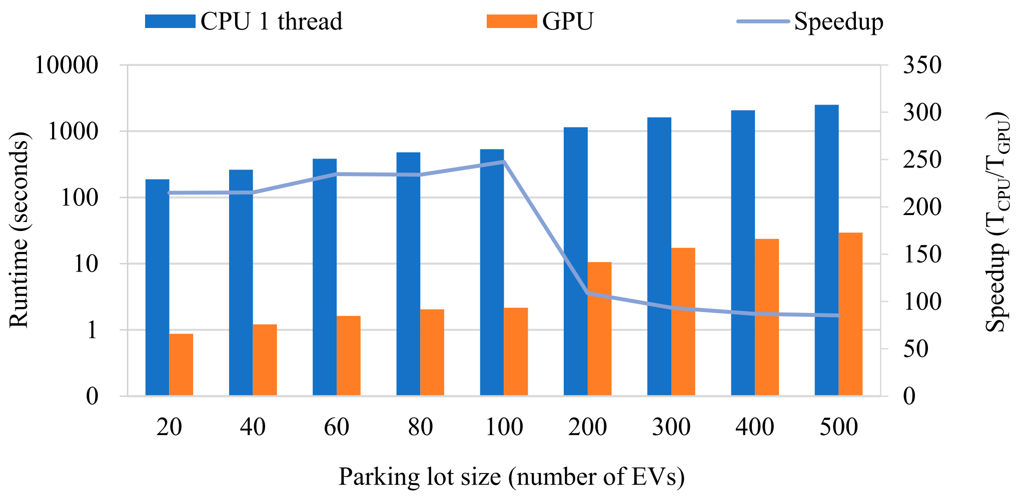


Figure 9. Runtime and speedup of the parallel implementation on GPU.

5. Discussion

This paper presented a parallel algorithm for the optimization of the EV recharge scheduling problem. The algorithm is based on the PSO but uses multiple runs to ensure the consistency of the quality of the solutions computed. A fitness function that encompasses the optimization objective and the constraints has been defined. To avoid the long execution time often inherent to metaheuristics, the algorithm was parallelized using two different approaches. The first approach uses task-level parallelization, which is suited for multicore CPUs. It ran the multiple PSO runs in parallel, reducing the execution time by a factor of 7.1x without affecting the end results. The second parallelization technique was data-level parallelization, which is suited for a GPU implementation, where all steps within the PSO were parallelized. By exploiting the massively parallel architecture of the GPU, this second technique led to a speedup of up to 247.6x compared to sequential execution on a CPU. The proposed algorithm was tested in simulation using nine different scenarios with parking lot sizes of 20 to 500 EVs.

Compared to previous works published in the literature, our proposed algorithm provided a better solution for the smallest parking lot size, but the main advantage is that our algorithm is scalable to large parking lot sizes, which are rarely used in other works. This was possible because the parallel algorithm on the GPU is so fast that it can be run with a very large number of candidate solutions and iterations while maintaining an acceptable runtime for real-time use. As an example, we were able to find solutions to the 20-EV parking lot in just 0.87 s and the 500-EV parking lot in 29.3 s. These times represent the minimum period at which the charging schedule could be updated on a continuous basis. Despite the important advantages of the proposed algorithm, one drawback needs to be mentioned. The PSO is a non-deterministic algorithm that can provide suboptimal results in any run. By proposing the multi-run PSO, we reduce the chances of obtaining suboptimal results, but the algorithm still remains non-deterministic, and there is always a possibility of obtaining a suboptimal result. One must be aware of this fact when using the proposed multi-run PSO.

In our future work, we intend to investigate the use of a hybrid algorithm where the multi-run PSO would be combined with a greedy algorithm for the problem of EV recharge scheduling. We also intend to focus on the problem of multi-parking lot EV scheduling, in which vehicles would be directed to a specific parking lot upon arrival based on the charging schedule of each lot.

6. Conclusions

The number of EVs in operation has increased drastically over the last few years and is expected to continue to increase in the future. EVs provide a significant environmental benefit but also a substantial burden on the power grid. One challenge that is of current interest to researchers is the EV recharge scheduling problem, where an optimized schedule is computed to recharge EVs within a smart parking lot. This paper presented a parallel multi-run PSO on a multicore CPU and GPU for the EV recharge scheduling problem. To better contextualize the contribution, we first presented a literature survey of previous methods published for this problem. We then presented the proposed method, which included the problem formulation, an introduction to the PSO and the GPU, the solution encoding, the fitness function developed, the task-level parallelization on the CPU, and the data-level parallelization on the GPU. We then tested the proposed parallel algorithm using nine scenarios of sizes ranging from 20 EVs to 500 EVs in five different tests. In the first test, we demonstrated that our multi-run PSO provides a better schedule with a lower recharge cost than previous solutions published in the literature, including other metaheuristics and greedy methods. In the second test, we showed that the proposed algorithm scales well to large parking lots with up to 500 EVs. In the third test, we show that the proposed multi-run PSO performs better than a single-run PSO for most scenarios with the same overall number of iterations. In the fourth and fifth tests, we showed the speedup achieved by the task-level parallelization on the multicore CPU and the data-level parallelization on the GPU. The speedups were 7.1x and 247.6x, respectively, compared to a sequential execution on a CPU. The runtime of the GPU implementation is 0.87 s for 20 EVs and 29.3 s for 500 EVs, which is small enough to allow for real-time scheduling. In conclusion, the parallel multi-run PSO algorithm proposed has the advantage of computing schedules with a lower overall recharge cost in a reduced execution time. This provides an economic advantage compared to not using any optimization method or using previously published optimization methods.

Supplementary Materials: The following supporting information can be downloaded at: https://www.mdpi.com/article/10.3390/electronics13091783/s1.

Author Contributions: Conceptualization, V.R., K.B. and M.T.; methodology, V.R., K.B. and M.T.; software, V.R., K.B. and M.T.; validation, V.R., K.B. and M.T.; formal analysis, V.R., K.B. and M.T.; investigation, V.R., K.B. and M.T.; resources, V.R., K.B. and M.T.; data curation, V.R., K.B. and M.T.; writing—original draft preparation, V.R., K.B. and M.T.; writing—review and editing, V.R., K.B. and M.T.; visualization, V.R., K.B. and M.T.; supervision, V.R., K.B. and M.T.; project administration, V.R., K.B. and M.T. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by NVIDIA Corp. who donated hardware components to conduct the simulation experiment.

Data Availability Statement: Data are contained within the article.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. International Energy Agency (IEA). World Energy Outlook 2022. Available online: https://www.iea.org/reports/world-energy-outlook-2022 (accessed on 28 March 2024).
2. Kalakanti, A.K.; Rao, S. Computational Challenges and Approaches for Electric Vehicles. *ACM Comput. Surv.* **2023**, *55*, 1–35. [CrossRef]

3. Muñoz, A.; Rubio, F. Evaluating genetic algorithms through the approximability hierarchy. *J. Comput. Sci.* **2021**, *53*, 101388. [CrossRef]
4. Amin, A.; Tareen, W.U.K.; Usman, M.; Ali, H.; Bari, I.; Horan, B.; Mekhilef, S.; Asif, M.; Ahmed, S.; Mahmood, A. A Review of Optimal Charging Strategy for Electric Vehicles under Dynamic Pricing Schemes in the Distribution Charging Network. *Sustainability* **2020**, *12*, 10160. [CrossRef]
5. Bitencourt, L.D.A.; Borba, B.S.M.C.; Maciel, R.S.; Fortes, M.Z.; Ferreira, V.H. Optimal EV charging and discharging control considering dynamic pricing. In Proceedings of the 2017 IEEE Manchester PowerTech, Manchester, UK, 18–22 June 2017; pp. 1–6. [CrossRef]
6. Martinenas, S.; Pedersen, A.B.; Marinelli, M.; Andersen, P.B.; Trreholt, C. Electric vehicle smart charging using dynamic price signal. In Proceedings of the 2014 IEEE International Electric Vehicle Conference (IEVC), Florence, Italy, 17–19 December 2014; pp. 1–6. [CrossRef]
7. Xu, P.; Li, J.; Sun, X.; Zheng, W.; Liu, H. Dynamic Pricing at Electric Vehicle Charging Stations for Queueing Delay Reduction. In Proceedings of the 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS), Atlanta, GA, USA, 5–8 June 2017; pp. 2565–2566. [CrossRef]
8. Mierau, M.; Kohrs, R.; Wittwer, C. A distributed approach to the integration of electric vehicles into future smart grids. In Proceedings of the 2012 3rd IEEE PES Innovative Smart Grid Technologies Europe (ISGT Europe), Berlin, Germany, 14–17 October 2012; pp. 1–7. [CrossRef]
9. Wu, H.; Pang, G.K.-H.; Choy, K.L.; Lam, H.Y. Dynamic resource allocation for parking lot electric vehicle recharging using heuristic fuzzy particle swarm optimization algorithm. *Appl. Soft Comput.* **2018**, *71*, 538–552. [CrossRef]
10. Nejati, S.A.; Chong, B.; Alinejad, M.; Abbasi, S. Optimal scheduling of electric vehicles charging and discharging in a smart parking-lot. In Proceedings of the 2021 56th International Universities Power Engineering Conference (UPEC), Middlesbrough, UK, 31 August–3 September 2021; pp. 1–6. [CrossRef]
11. Zheng, K.; Teng, S.; Zhao, Y. Research on Optimal Scheduling Strategy for Electric Vehicle Charging and Discharging Based on Time-of-Use Electricity Price. In Proceedings of the 2023 6th Asia Conference on Energy and Electrical Engineering (ACEEE), Chengdu, China, 21–23 July 2023; pp. 519–524. [CrossRef]
12. Tan, M.; Zhang, Z.; Ren, Y.; Richard, I.; Zhang, Y. Multi-Agent System for Electric Vehicle Charging Scheduling in Parking Lots. *Complex Syst. Model. Simul.* **2023**, *3*, 129–142. [CrossRef]
13. Abdullah-Al-Nahid, S.; Aziz, T. A novel day ahead charging scheme for electric vehicles with time of use-based prioritization supported by genetic algorithm. In Proceedings of the 2022 Global Energy Conference (GEC), Batman, Turkey, 26–29 October 2022; pp. 19–24. [CrossRef]
14. Wolpert, D.H.; Macready, W.G. No free lunch theorems for optimization. *IEEE Trans. Evol. Comput.* **1997**, *1*, 67–82. [CrossRef]
15. Loscos, D.; Martí-Oliet, N.; Rodríguez, I. Generalization and completeness of stochastic local search algorithms. *Swarm Evol. Comput.* **2022**, *68*, 100982. [CrossRef]
16. Kizil, A.; Karabulut, K. Effects of Parameters of an Island Model Parallel Genetic Algorithm for the Quadratic Assignment Problem. In Proceedings of the 2019 8th International Congress on Advanced Applied Informatics (IIAI-AAI), Toyama, Japan, 7–11 July 2019; pp. 444–449. [CrossRef]
17. Roberge, V.; Labonté, G.; Tarbouchi, M. Minimizing Fuel Consumption for Surveillance Unmanned Aerial Vehicles Using Parallel Particle Swarm Optimization. *Sensors* **2024**, *24*, 408. [CrossRef] [PubMed]
18. Lalwani, S.; Sharma, H.; Satapathy, S.C.; Deep, K.; Bansal, J.C. A Survey on Parallel Particle Swarm Optimization Algorithms. *Arab. J. Sci. Eng.* **2019**, *44*, 2899–2923. [CrossRef]
19. Roberge, V.; Tarbouchi, M.; Labonté, G. Fast Genetic Algorithm Path Planner for Fixed-Wing Military UAV Using GPU. *IEEE Trans. Aerosp. Electron. Syst.* **2018**, *54*, 2105–2117. [CrossRef]
20. Ozdemir, M. Particle Swarm Optimization for Continuous Function Optimization Problems. *Int. J. Appl. Math. Electron. Comput.* **2017**, *5*, 47–52. [CrossRef]
21. Chen, C.-R.; Chen, Y.-S.; Lin, T.-C. Optimal Charging Scheduling for Electric Vehicle in Parking Lot with Renewable Energy System. In Proceedings of the 2019 IEEE International Conference on Systems, Man and Cybernetics (SMC), Bari, Italy, 6–9 October 2019; pp. 1684–1688. [CrossRef]
22. Amiri, S.S.; Jadid, S. Optimal charging schedule of electric vehicles at battery swapping stations in a smart distribution network. In Proceedings of the 2017 Smart Grid Conference (SGC), Tehran, Iran, 20–21 December 2017; pp. 1–8. [CrossRef]
23. Ibrahim, A.A.; Wahid, S.S.A.; Mohamed Kamari, N.A.; Mohd Zaman, M.H.; Zulkifley, M.A. Optimal Scheduling of Plug-in Electric Vehicles Using Binary Gravitational Search Algorithm with A Suitable Decision Function. In Proceedings of the 2022 IEEE 20th Student Conference on Research and Development (SCOReD), Bangi, Malaysia, 8–9 November 2022; pp. 96–100. [CrossRef]
24. Pan, K.; Liang, C.-D.; Lu, M. Optimal scheduling of electric vehicle ordered charging and discharging based on improved gravitational search and particle swarm optimization algorithm. *Int. J. Electr. Power Energy Syst.* **2024**, *157*, 109766. [CrossRef]
25. Rho, S.; Chae, M.; Won, D. Forecast-based Optimal Operation of EV Charging Station with PV Considering Charging Demand and Distributed System. In Proceedings of the 2024 IEEE Power & Energy Society Innovative Smart Grid Technologies Conference (ISGT), Washington, DC, USA, 19–22 February 2024; pp. 1–5. [CrossRef]
26. Zhang, D.; Qian, L.; Mao, B.; Huang, C.; Huang, B.; Si, Y. A Data-Driven Design for Fault Detection of Wind Turbines Using Random Forests and XGboost. *IEEE Access* **2018**, *6*, 21020–21031. [CrossRef]

27. Wu, H.; Pang, G.K.H.; Choy, K.L.; Lam, H.Y. A scheduling and control system for electric vehicle charging at parking lot. In Proceedings of the 2017 11th Asian Control Conference (ASCC), Gold Coast, Australia, 17–20 December 2017; pp. 13–18. [CrossRef]
28. Abdullah, E.A.; Ahmed Saleh, I.; Al Saif, O.I. Performance Evaluation of Parallel Particle Swarm Optimization for Multicore Environment. In Proceedings of the 2018 International Conference on Advanced Science and Engineering (ICOASE), Duhok, Iraq, 9–11 October 2018; pp. 81–86. [CrossRef]
29. Roberge, V.; Tarbouchi, M.; Labonté, G. Comparison of Parallel Genetic Algorithm and Particle Swarm Optimization for Real-Time UAV Path Planning. *IEEE Trans. Ind. Inform.* **2013**, *9*, 132–141. [CrossRef]
30. Roberge, V.; Tarbouchi, M.; Allaire, F. Parallel Hybrid Metaheuristic on Shared Memory System for Real-Time UAV Path Planning. *Int. J. Comput. Intell. Appl.* **2014**, *13*, 1450008-1–1450008-16. [CrossRef]
31. Abdelatti, M.; Sodhi, M.; Sendag, R. A Multi-GPU Parallel Genetic Algorithm For Large-Scale Vehicle Routing Problems. In Proceedings of the 2022 IEEE High Performance Extreme Computing Conference (HPEC), Virtual, 19–23 September 2022; pp. 1–8. [CrossRef]
32. Benaini, A.; Berrajaa, A. Parallel genetic algorithm on GPU for the robust uncapacitated single allocation p-hub median problem with discrete scenarios. In Proceedings of the 2020 5th International Conference on Logistics Operations Management (GOL), Rabat, Morocco, 28–30 October 2020; pp. 1–8. [CrossRef]
33. Beni, M.S.; Sojoodi, A.H.; Khunjush, F. A GPU-Enabled Extension for Apache Ignite to Facilitate Running Genetic Algorithms. In Proceedings of the 2020 20th International Symposium on Computer Architecture and Digital Systems (CADS), Rasht, Iran, 19–20 August 2020; pp. 1–8. [CrossRef]
34. Ohira, R.; Islam, M.S. GPU Accelerated Genetic Algorithm with Sequence-based Clustering for Ordered Problems. In Proceedings of the 2020 IEEE Congress on Evolutionary Computation (CEC), Glasgow, UK, 19–24 July 2020; pp. 1–8.
35. CUDA Toolkit. NVIDIA Developer. Available online: https://developer.nvidia.com/cuda-toolkit (accessed on 3 April 2024).
36. Kennedy, J.; Eberhart, R. Particle swarm optimization. In Proceedings of the IEEE International Conference on Neural Networks, Perth, Australia, 27 November–1 December 1995; Volume 4, pp. 1942–1948. [CrossRef]
37. The OpenMP. API Specification for Parallel Programming. OpenMP. Available online: https://www.openmp.org/ (accessed on 26 March 2024).
38. Clerc, M.; Kennedy, J. The particle swarm—Explosion, stability, and convergence in a multidimensional complex space. *IEEE Trans. Evol. Comput.* **2002**, *6*, 58–73. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.