Computers and Electrical Engineering 92 (2021) 107085

Contents lists available at ScienceDirect

Computers and Electrical Engineering

journal homepage: www.elsevier.com/locate/compeleceng

Parallel Quadri-valent Quantum-Inspired Gravitational Search Algorithm on a heterogeneous platform for wireless sensor networks☆,☆☆

Mina Mirhosseini[a], Mahmood Fazlali[a,*], Hadi Tabatabaee Malazi[b], Sayyed Kamyar Izadi[c], Hossein Nezamabadi-pour[d]

[a] Department of Computer and Data Sciences, Faculty of Mathematical Sciences, Shahid Beheshti University, Tehran, Iran
[b] School of Computer Science and Statistics, Trinity College Dublin, Dublin, Ireland
[c] Department of Computer Science, Faculty of Mathematical Sciences, Alzahra University, Tehran, Iran
[d] Department of Electrical Engineering, Shahid Bahonar University of Kerman, Kerman, Iran

ARTICLE INFO

Keywords:
Quantum computing
Heterogeneous platform
Compute unified device architecture
Open-MP
Graphics processing unit

ABSTRACT

Sensor nodes in a wireless sensor Network are assigned for different operational modes to perfume application-specific objectives. The decision to assign operational modes to nodes is a challenging problem in the presence of multiple criteria including energy-efficient, maintaining network connectivity, and fulfilling application goals. Several metaheuristic methods are introduced in the literature to address this NP-hard problem, however, these methods require further improvements in execution-time and finding the optimum solution. In this research, we propose an improved version of a metaheuristic method called Quadri-valent Quantum-Inspired Gravitational Search Algorithm (QQIGSA) to solve Quadri-valent problems by applying a *Not Q-Gate* and paralleling QQIGSA method on the graphics processing unit. The proposed method employs a heterogeneous platform and justifies its parameters. The experimental results show that the performance enhancement from 1.8 to 2.25 compared to the previous parallel implementations. Moreover, we achieve the speedup of 8 by using the proposed heterogeneous paralleling technique.

1. Introduction

Wireless sensor networks (WSN) play an influential role in different areas such as smart city, industry 4.0 based factories, precision agriculture, and smart hospitals. They provide valuable information by sensing the surrounding environment and sending them toward a base-station for further analysis and actions. The design of a WSN is a challenging task for various reasons. First, the life cycle of these resource-constrained sensor nodes is tightly coupled with their energy consumption which raises energy efficiency as one of the most important design aspects. Second, the network has to maintain the connectivity of all the nodes to prevent network partitioning. Finally, WSN applications may assign specific modes (e.g., cluster head, coordinator) to sensor nodes according to their mission. Although considering these aspects (i.e., energy efficiency, network connectivity, and application-specific roles) makes a WSN design a challenging and complex task, it boosts their performance and dependability [1,2].

☆ This paper was recommended for publication by associate editor M. Malek.
☆☆ This paper is for CAEE special section VSI:pdcat3. Reviews processed and recommended for publication to the Editor-in-Chief by Guest Editor Dr. Hui Tian.
* Corresponding author.
E-mail addresses: mi_mirhosseini@sbu.ac.ir (M. Mirhosseini), fazlali@sbu.ac.ir (M. Fazlali), TabatabH@tcd.ie (H. Tabatabaee Malazi), k.izadi@alzahra.ac.ir (S.K. Izadi), nezam@uk.ac.ir (H. Nezamabadi-pour).

https://doi.org/10.1016/j.compeleceng.2021.107085
Received 30 March 2020; Received in revised form 5 March 2021; Accepted 8 March 2021
Available online 25 March 2021

To model the assignment of different operation modes to sensor nodes, we consider a network area of the size $r \times c$, where all sensors are deployed in a grid shape (on all $r \times c$ junctions). These sensor nodes are identical and are capable to be in one of the four supported operational states, including inactive (Off), Cluster Head (CH), the sensor with Low Signal Range (LSR), or the sensor with a High Signal Range (HSR). The problem is to assign a proper operational mode to these nodes while maintaining the network connectivity, specific-application requirements, and energy efficiency. In other words, the problem is to find the operation modes of the sensors at each point in time, like a dynamic sequence, to increase the network lifetime according to the number of sensing/data collection cycles. Therefore, the algorithm needs to run repeatedly while dynamically considering the latest energy level of the sensors after each cycle [1]. This problem is reported as NP-hard [1] and metaheuristic algorithms are considered as potential approaches to solve it.

Metaheuristic algorithms are population-based and non-deterministic optimization approaches that address the computability challenge of NP-hard problems. They are successful methods that can provide near-optimum solutions in a reasonable time for complex and NP-hard problems. Genetic Algorithms (GA), Simulated Annealing (SA), Ant Colony Optimization (ACO), Particle Swarm Optimization (PSO), Imperialist Competitive Algorithm (ICA), and Gravitational Search Algorithm (GSA) are some of the well-known metaheuristic algorithms that are used frequently in different applications [3].

Although metaheuristic methods can solve NP-hard problems faster compared to the traditional deterministic methods, in some cases, they suffer from long computation time. Another limitation of metaheuristic methods is that by increasing the search space of the problem, they may not converge toward finding the sub-optimal or optimal solutions. In some cases, increasing the size of the population may enhance the quality of solutions but, it can lead to increasing the running time. To mitigate this limitation, the parallel execution of metaheuristic methods are suggested by researchers that helps to enhance the quality of the solutions as well as reducing the run-time. Considering both the Central Processing Unit (CPU) and Graphics Processing Unit (GPU) as the processing units in a computer and their unique advantages and limitations, the collaboration between CPU and GPU can provide a high-performance computing environment. A combination of CPU and GPU platforms creates a new platform known as a Heterogeneous Computing (HC) platform which has higher computational gains than multi-core systems. This platform aims to employ task paralleling on CPU and data paralleling on GPU. Therefore, many researchers are motivated to use this heterogeneous computing technique in different applications. *Mittal* et al. [4] present an exhaustive survey of heterogeneous computing approaches.

Several works introduce solutions to the aforementioned WSN problem using parallel metaheuristic methods. A Binary Genetic Algorithm (BGA) is developed by *Ferentinos* et al. [1] to solve the problem in WSNs. *Hojjatoleslami* et al. [2] use Binary Particle Swarm Optimization (BPSO) to address the problem. A modified version of BQIGSA is developed and adopted for the optimal design of WSN in [5]. The current problem is a problem with four states; so, a new version of the quantum GSA is called Quadri-valent Quantum-Inspired Gravitational Search Algorithm (QQIGSA). It is suitable for four-state problems or Quadri-valent encoded problems which is proposed in [6]. This approach has better achievements compared to BGA [1] and BPSO [2]. It is worth noting that to enhance the speed of this algorithm, a parallel implementation of improved QQIGSA, using Open-MP was suggested in [7].

This paper is an extension of work [7] initially presented in TopHPC 2019. In [7], the speed of the improved QQIGSA (IQQIGSA) is increased by paralleling some part of the algorithm, using a multi-core CPU. In this paper, we offer the paralleling technique using the CPU–GPU heterogeneous to have a higher speed-up on the execution time of the algorithm. This lets us increase the population size which leads to the improvement of the quality of results. Moreover, in this paper, parameters related to the proposed IQQIGSA have been justified. Besides, in this paper following the previous works [1,2,5,6], an improved version of QQIGSA [6] is developed by applying a *Not Q-Gate*. This work leads to an increase in the diversity of the population to find better results. To deal with a large searching space, we require a larger population to find more acceptable accuracy of the results. However, based on the limitation of resources, we may not be able to satisfy this requirement. Furthermore, for problems that need more complex computations, the running time of the algorithm may increase and may become intolerable. Therefore, we use the parallelism technique to accelerate the speed and enhance the accuracy of the method.

We propose applying a heterogeneous platform for parallel processing using both CPUs and GPUs to accelerate the speed of the proposed improved QQIGSA. The main problem is divided into three partial problems; one of them is executed on the CPU, and the others are executed on the GPU. The proposed heterogeneous parallelism technique of the QQIGSA can enhance the speed and the quality of results in the WSN problem. The main contributions of this paper are summarized as:

- A modified version of the QQIGSA is proposed by applying a *Not Q-Gate*.
- The main parameters of the suggested IQQIGSA are justified using the Taguchi method.
- The proposed IQQIGSA uses a parallelism technique based on the CPU–GPU heterogeneous technique to increase the speed of the algorithm.
- The algorithm is developed to handle the design of WSNs with four operational states of sensors.

The performance and the speed of the parallel and improved QQIGSA in the WSN optimization problem are investigated, and a comprehensive comparison between state-of-the-art ones including BGA [1], BPSO [2], the improved BQIGSA [5], and QQIGSA [6] are performed. Moreover, the dynamic optimal design algorithm is applied. The results show that the proposed method surpasses the previous approaches in terms of speed, accuracy, and the lifespan of the network as well as the base station energy consumption.

The rest of this paper is organized as follows. Section 2 describes related work followed by preliminaries in Section 3. Section 4 presents the suggested algorithm and, two paralleling techniques. Section 5 gives the experimental results of adapting the proposed algorithm to the WSN problem, and Section 6 concludes the paper.

2. Related works

Metaheuristic methods potentially can solve NP-hard problems in an acceptable time compared to exact methods. However, in some cases, they may suffer from the time-consuming exploration of large solution spaces. To mitigate the limitation, paralleling metaheuristic methods can be an option to decrease the execution time. *Crainic* [8] presents a comprehensive review of parallel metaheuristic methods. It is considered that some parallel metaheuristic methods can achieve super-linear performance.

Several metaheuristic methods are provided by researchers to solve the described problem of optimal design of a WSN. *Ferentinos* et al. [1] used a binary genetic algorithm (BGA) to solve this problem. *Hojjatoleslami* et al. [2] introduced the binary particle swarm optimization (BPSO) method to address the problem, however, the quality of their results requires more improvements. To improve the quality, a modified version of BQIGSA [5] and QQIGSA [6] were devised by the researchers. Later, to increase the speed of the QQIGSA, a parallel implementation using Open-MP was suggested in [7]. The work of this paper is a continuation of our previous work in [7] to accelerate the speed of the algorithm.

Gravitational Search Algorithm (GSA) is the metaheuristic base in our work. Two parallel versions of GSA by GPU implementation are presented in [9] and [10]. GSA is one of the recently introduced metaheuristic methods with acceptable exploration ability. This algorithm is inspired by the motion and gravity laws of Newtonian [11]. Recently, several quantum versions of GSA suitable for binary-valued problems are introduced in [12,13]. The comparative results indicate that BQIGSA [12] can be considered as one of the most efficient methods in solving the binary optimization problem. Furthermore, the Quadri-valent version of quantum-inspired GSA called QQIGSA was proposed in [6] for dealing with Quadri-valent problems.

In GSA, only one parameter must be adjusted which is an advantage compared to other metaheuristic methods that have multiple adjustable parameters. Gravitational constant or G_0 has an important role in GSA and its versions. Several works are published to address how to adjust this constant. For example, *Ibrahim* et al. [14] use chaotic neural oscillators (CNO) to set the gravitational constant. In [15], Aggregative Learning Gravitational Search Algorithm (ALGSA) and in [16], Hierarchical GSA (HGSA) are suggested to address the problems of premature convergence and low search performance of GSA. In ALGSA, each particle adapts its own gravitational constant to enhance the search performance. Also, HGSA studies the problem from the population topology viewpoint and apply an effective gravitational constant.

Another method to improve the performance of GSA is proposed in [17]. This method, which is called chaotic gravitation search algorithms, combines the GSA and chaos to take advantages of both techniques. To balance the exploitation and exploration abilities, a self-adaptive method is developed in [18] for setting the gravitation constant. Some other methods can be found in [19].

One of the paralleling methods frequently used by researchers is to apply the multi-core platform and Open-MP to implement the algorithms. Open-MP supports parallel programming on shared-memory multiprocessor systems. Numerous algorithms are converted to parallel using Open-MP to obtain acceptable speedup. This is besides its simple implementation in comparison with other parallelism techniques such as Compute Unified Device Architecture (CUDA) programming implemented on GPU. Moreover, Open-MP gives results with an accuracy equivalent to the sequential implementation [20].

3. Preliminaries

3.1. CUDA for GPU and Open-MP for multicore systems

Compute Unified Device Architecture (CUDA) is a programming model used for parallel programming on many-core NVIDIA GPUs. Here, the problem is divided into sub-problems, which are computed using blocks of threads in parallel. A block is defined as a batch of threads that can collaborate by synchronizing and sharing data via shared memory. The instructions of CUDA are implemented in a function called, Kernel.

Fig. 1 shows a Kernel that is called by defining a grid of blocks and the block is specified by a unique pair of indices (in the case of two-dimensional). Moreover, threads in blocks can be organized as a two-dimensional grid with unique indices set. All threads involved in a kernel have access to global memory. Furthermore, each thread has several registers with limitations in the size, and to cooperate with the threads of a block, they have to access through a fast shared memory [10,21].

It is worth noting that Open-MP is a library for the implementation of parallel programs for shared memory multi-core processors. It is composed of a set of instructions that can be embedded in a program written by C, C++, or Fortran languages. Recently, many programs are implemented in parallel using Open-MP, achieving proper speedup in addition to its simple implementation [22,23]

3.2. Preliminaries on QQIGSA

QIEAs or Quantum-inspired evolutionary algorithms are a class of metaheuristic methods that are produced by a combination of evolutionary algorithms and quantum computing. QQIGSA proposed in [7] is made by merging GSA [11] and quantum computing to solve Quadri-valent problems. This section describes the procedure of QQIGSA.

In QQIGSA, each Qubit is composed of a pair of (α, β) which is capable to be found in the states "0", "1", "2", "3" and even in a combination of them at the same time. $|\alpha|^2$ and $|\beta|^2$ determine the chance of finding the Qubit in each state. In QQIGSA, an agent is demonstrated as a vector composed of n Qubits (Eq. (1)).

$$qb_i(t) = [qb_i^1(t), qb_i^2(t), \dots, qb_i^n(t)] = \begin{bmatrix} \alpha_i^1(t) & \alpha_i^2(t) & \dots & \alpha_i^n(t) \\ \beta_i^1(t) & \beta_i^2(t) & \dots & \beta_i^n(t) \end{bmatrix} \qquad (1)$$

$$i = 1, 2, \dots, Npop$$

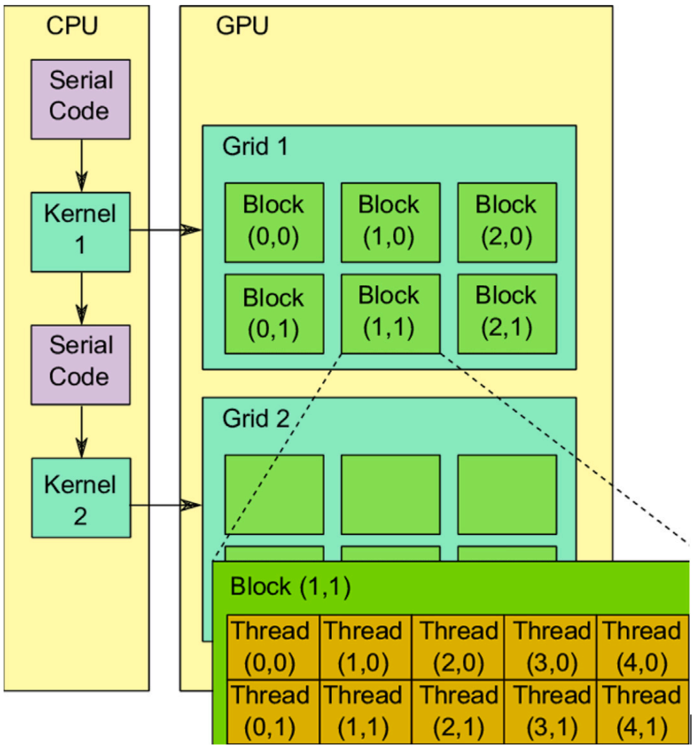

Fig. 1. Illustration of the GPU architecture [10].

where n is the dimension size of search space which depends on the problem, $Npop$ is the size of the population and each pair of $\begin{bmatrix}\alpha_i^d(t)\\ \beta_i^d(t)\end{bmatrix}$ should meet Eq. (2).

$$|\alpha_i^d|^2 + |\beta_i^d|^2 = 1 \tag{2}$$

Firstly, an initial population $Q(t)$, consisting of $Npop$ objects in an n-dimensional searching space, is generated randomly. Then, the observation function is performed using Alg. 1 to collapse the quantum state of Q-bits into a single value. This process makes $SW(t) = \{X_1(t), X_2(t), \dots, X_{Npop}(t)\}$, in which $X_i(t) = \left(x_i^1(t), x_i^2(t), \dots, x_i^n(t)\right)$.

Algorithm 1: Observation function

```
if (rand [0,1] < $\left(\alpha_i^d(t)\right)^2$) then
    if ($\alpha_i^d(t) < 0$) then
        $x_i^d(t) = 0$ ;
    else
        $x_i^d(t) = 2$ ;
    end if
else
    if ($\beta_i^d(t) < 0$) then
        $x_i^d(t) = 1$ ;
    else
        $x_i^d(t) = 3$ ;
    end if
end if
```

Later, the fitness amount of each solution $X_i(t)$ is evaluated. The set

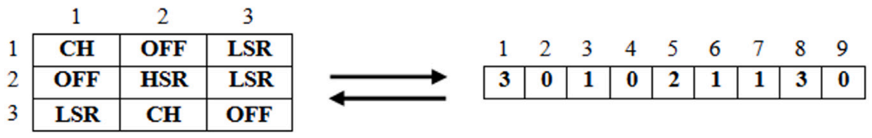

Fig. 2. Representation as a vector (on the right), and WSN design (on the left) [7].

$SB(t) = \{B_1(t), B_2(t), \ldots, B_{Npop}(t)\}$ where, $B_i(t) = \left(b_i^1(t), b_i^2(t), \ldots, b_i^n(t)\right)$, collects the best solutions found during the iterations t. If $X_i(t)$ has a better fitness value, the $B_i(t) \in SB(t)$ is replaced by $X_i(t) \in SW(t)$; regarding $SB(0) = SW(0)$. Then, the mass $M_i(t)$ for $i = 1, 2, \ldots, Npop$ is computed for solutions collected in $SB(t)$, using Eq. (3).

$$M_i(t) = \frac{fit_i(t) - worst(t)}{\sum_{j=1}^{Npop}(fit_j(t) - worst(t))} \tag{3}$$

Then, the angular velocity specifies the movement value toward "0", "1", "2" or "3" by applying the rotational quantum gate (*RQ-gate*) on Qubits. The angular velocity of each Q-bit is calculated using Eqs. (4) and (5).

$$\omega_i^d(t+1) = rand_i \times \omega_i^d(t) + a_i^d(t) \tag{4}$$

$$a_i^d(t) = \sum_{j \in k_{best}} G(t) \frac{M_j(t)}{R_{ij}(t) + \epsilon} \left(b_j^d(t) - x_i^d(t)\right) \tag{5}$$

In these relations, $\omega_i^d(t)$ and $a_i^d(t)$ respectively denote the angular velocity and acceleration of object i in dimension d at iteration t. Also, $G(t)$ refers to the gravitational constant in iteration t, which initialized to G_0 at the first iteration and is decreased during iterations as $G(t) = G_0(1 - 0.95\frac{t}{ITE})$, in which ITE is the iteration number of the procedure. Moreover, $M_j(t)$ is the mass of object j at iteration t. The set of k_{best} includes k best-found solutions in $SB(t)$. The size of k_{best} is linearly decreased by iterations. $x_i^d(t)$ and $b_j^d(t)$ are respectively the value of solutions $X_j(t) \in SW$ and $B_j(t) \in SB(t)$ at dimension d in iteration t. $R_{ij}(t)$ computes the distance between objects i and j as Eq. (6). Also, ϵ is a very small value to prevent dividing into zero.

$$R_{ij}(t) = \frac{\sum_{l=1}^{n} |x_i^l(t) - x_j^l(t)}{3 \times n} \tag{6}$$

The rotation angle of Qubit i at dimension d is calculated by Eq. (7). Then, each Qubit is updated independently using Eq. (8). Moreover, the iteration t is increased by one. The procedure is repeated until the condition of a predefined number of iterations is met [7].

$$\Delta\theta_i^d = \begin{cases} -\omega_i^d(t+1), & if\,|b_j^d(t) - x_i^d(t)| = 3 \\ \omega_i^d(t+1), & if\,|b_j^d(t) - x_i^d(t)| = 1 \\ 0, & otherwise \end{cases} \tag{7}$$

$$\begin{bmatrix} \alpha^d(t+1) \\ \beta^d(t+1) \end{bmatrix} = \begin{bmatrix} \cos(\Delta\theta^d) & \sin(\Delta\theta^d) \\ -\sin(\Delta\theta^d) & \cos(\Delta\theta^d) \end{bmatrix} \begin{bmatrix} \alpha^d(t) \\ \beta^d(t) \end{bmatrix} \quad d = 1, 2, \ldots, n \tag{8}$$

3.3. Problem of optimal design for WSNs in precision agriculture

WSNs have the ability of sensing, sending, and managing data. In a precision agriculture application, these data include temperature, humidity, soil moisture, and solar radiation to provide precise farm management. Here, the approach based on QQIGSA is described to design a WSN topology to enhance the lifetime of the network while the connectivity constraints, energy-related, and specific application parameters are met. underneath, it is described how to encode objects, and how to formulate the fitness function to evaluate objects.

3.3.1. Representation of object

The design objective of a WSN is to assign the sensors' operational states (including inactive (Off), cluster head (CH), the sensor with low signal range (LSR), and the sensor with high signal range (HSR)) so that the fitness value is maximized. In the proposed IQQIGSA, each WSN design is considered as a possible solution encoded as a vector composing the operational state of sensors. The size of the vector is equal to the number of located sensors on the area. The elements of the vector can get the value 0 for OFF, 1 for LSR, 2 for HSR, or 3 for CH operational states. Fig. 2 shows As an example of a WSN design and its representation as a vector.

3.3.2. Fitness evaluation

A fitness function is applied to evaluate the quality of each solution in a WSN design. In this application, three sets of parameters are considered including connectivity, energy-related, and application-specific parameters. These parameters that have to be minimized, are combined to form a single objective function. This weighted fitness function is defined by Eq. (9) where, w_1, w_2, …and w_7 are the weighting coefficients indicating the importance of each parameter.

$$fit = \frac{1}{(w_1.MRD + w_2.SDE + w_3.SCE + w_4.SORE + w_5.OE + w_6.CE + w_7.BCP)} \tag{9}$$

In this function, MRD, SDE and SCE are application-specific parameters, referring to uniformity of nodes, Spatial Density error, and Sensors-per-Cluster-head Error, respectively. Moreover, SORE or "Sensor-Out-of-Range Error" is a connectivity parameter. This parameter, which depends on the sensor's signal board, ensures that each active sensor can communicate with its related cluster head. OE or *Operational Energy* as an energy-related parameter computes the energy consumed by active sensors along with a time. It highly depends on the operational state of sensors. CE or *Communication Energy* is another energy-related parameter. This parameter shows the energy consumed for communication between active sensors and their related CH. Parameter BCP or *Battery Capacity Penalty* as an energy-related parameter with an essential role in increasing the lifetime of the WSN. For more details regarding these parameters, the interested readers may refer to [1,2,5,6].

4. The proposed improved and parallel QQIGSA

QQIGSA is a QIEA well suitable for Quadri-valent optimization problems. To enhance the performance of this algorithm and avoid local optimum solutions, it is necessary to allow the algorithm to make the population and have new unprecedented solutions. Also, a large number of function evaluations and algorithm operators results in high CPU execution time. Therefore, an attempt to accelerate QQIGSA is necessary.

In this paper, the performance of the QQIGSA is enhanced by adding a new quantum operator namely *Not Q-Gate*. The effect of this operator in QQIGSA is similar to the effect of the mutation in the genetic algorithm to avoid the local optimum and introduce diversity to the population. The *Not Q-Gate* avoids minimum/ maximum by avoiding the population from getting too similar to each other. This reason also explains that IQQIGSA may avoid taking only the fittest solutions in making the next population by adding *Not Q-Gate*. It is rather a semi-random selection toward the fitter solutions.

Not Q-Gate selects one or more objects from the current population with a predefined probability and then alters its present state. It can lead to a complete change of the object. In this case, the diversity of the population is increased and may lead to finding more optimal solutions. This change can occur in all or some iterations of the algorithm according to the user-defined probability. The value of this probability should be set to a low value between 0 and 1, otherwise, the algorithm may become a primitive random search. A typical approach for implementing *Not Q-Gate* can be selecting an object and changing its α and β. If object i is selected randomly, this object is turned from $qb_i(t) = \begin{bmatrix} \alpha_i^1(t) & \alpha_i^2(t) & \dots & \alpha_i^n(t) \\ \beta_i^1(t) & \beta_i^2(t) & \dots & \beta_i^n(t) \end{bmatrix}$ to $qb_i(t) = \begin{bmatrix} \beta_i^1(t) & \beta_i^2(t) & \dots & \beta_i^n(t) \\ \alpha_i^1(t) & \alpha_i^2(t) & \dots & \alpha_i^n(t) \end{bmatrix}$. It helps to promote the diversity in QQIGSA and may preserve the algorithm from the local optimum.

The experimental results indicate that the suggested IQQIGSA can produce better solutions to solve the optimal WSN design issue, compared to the BGA [1], BPSO [2], QQIGSA [6] and improved BQIGSA [5] but it still needs an accelerate in the execution speed. After executing the algorithm on the network and determining the optimal operational modes of sensors, the battery charges of sensors are changed, and the program has to be executed again on WSN with updated battery charges in the next measuring cycle. So, the execution time of the algorithm needs to be decreased to have fast decisions.

Parallelism is an efficient approach to decrease the running time of the algorithm. For this purpose, we use two approaches. The first is using Open-MP programming, and the other is using both Open-MP and CUDA programming. By experimenting with the current WSN problem, we identify that re-evaluating the fitness of agents is the most time-consuming section. Moreover, computing the fitness function is well suitable to be run on a multi-core platform, and the other sections involving matrix computations are suitable for running on GPU.

Parallel QQIGSA by Open-MP is down by partitioning the population into sub-populations and computing the fitness function of objects independently. In each iteration, all objects are separate from each other and can be easily computed in parallel. It is worth noting that no communication between cores is required [7].

The pseudocode of the suggested IQQIGSA is converted to parallel for Open-MP and is presented in Alg. 2. The only difference between the sequential code of QQIGSA and IQQIGSA is adding the *pragma* statement, containing the private and shared variables, before the for-loop, and applying the *Not Q-Gate* at the end. The *pragma* statement instructs Open-MP to the parallel execution of the *for-loop* block.

The other approach is to compute the fitness values by multi-core platform and compute other sections by GPU. This can be implemented by assigning one block to one agent. All threads should be able to retrieve the position, Q-bit, angular velocity, and acceleration of agents with a unique pair as $[AgentID, DimensionID]$. Threads of block access to the dimension values using an index by $AgentID \times n + DimensionID$ where $AgentID$ is $blockId.x$ and $DimensionID$ is $threadId.x$.

Alg. 3 presents the pseudocode of the proposed and parallel algorithm by CPU–GPU heterogeneous computing technique. Each step is described in the following. Moreover, Fig. 3 shows the data and vector organization in global memory in the GPU. For matrix in the size $Npop \times n$, each agent is assigned to a block and each dimension is assigned to a thread which is demonstrated by an arrow. Besides, in this figure the expression in the form of $\lll Gs, Bs \ggg$ between the kernel name and the parenthesized arguments

Algorithm 2: The pseudocode of the proposed improved and parallel QQIGSA using Open-MP

```
1: Input: $ITE$, $N$, $p_N$, $G_0$;
2: Output: a topology from $SB$ with the best fitness value
3: $t = 0$;
4: $SB(t) = \{\}$;
5: Initialize Q(t);
6: while ($t < ITE$) do
7:    Make $SW(t)$ by observing $Q(t)$ using Alg. 1
8:    #Pragma for each object i {run in parallel}
9:        Evaluate $SW(t,i)$ by Eq. (9);
10:       Updating SB(t,i);
11:   Update_mass $M_i(t)$, i=1,2,...,N by Eq. (3);
12:   Update_Qbit Q(t) by RQ-gate to make $Q(t+1)$ by Eqs. (4), (5), (7), (8);
13:   Apply Not Q-Gate on $Q(t+1)$ by probability $p_N$
14:   t=t+1;
15: end while
```

Algorithm 3: The pseudocode of the proposed improved and parallel QQIGSA using Open-Mp

```
1: Input: $ITE$, $N$, $p_N$, $G_0$;
2: Output: a topology from $SB$ with the best fitness value
3: $t = 0$;
4: $SB(t) = \{\}$;
5: Initialize $\lll Npop, n \ggg (Q(t), V(t))$;
6: while ($t < ITE$) do
7:    Observation $\lll Npop, n \ggg (Q(t), SW(t))$ using Alg. 1
8:    Pragma for each agent i {run in parallel}
9:        Evaluate $SW(t,i)$ by Eq. (9);
10:   Updating_SB$\lll Npop, n \ggg (fitness(t), SW(t), SB(t))$;
11:   Update_Mass$\lll 1, n \ggg (Fitness(t), M(t))$ by Eq. (10).
12:   Update_distances$\lll Npop \times Npop, n \ggg (SB(t), R(t))$ using Eq. (6)
13:   Update_Velosity$\lll Npop, n \ggg (SW(t), SB(t), R(t), M(t), G(t), V(t))$ using Eqs. (4), (5)
14:   Update_Qbits$\lll Npop, n \ggg (Q(t), V(t), SB(t), SW(t), Q(t+1))$ by Eqs. (7), (8)
15:   Not_Q-Gate$\lll 1, n \ggg (Q(t+1), p_N)$
16:   cudaTreadSynchronize();
17:   t=t+1;
18: end while
```

shows that Gs is the grid size or the number of blocks, and Bs specifies the block size or the number of threads per block. Both Gs and Bs are of type $dim3$.

Initialization step: Initialize $\lll Npop, n \ggg Q(t), V(t)$: In this step, the Q-bits of all agents assigned randomly satisfying Eq. (2). $Q(t)$ is a 3D matrix in the size of $Npop \times n \times 2$, we split that into two 2D matrices $Q_\alpha(t)$ and $Q_\beta(t)$, both in the size of $Npop \times n$. Firstly, $Q_\alpha(t)$ and $Q_\beta(t)$ get random numbers and they are normalized to get a value between $[-1, 1]$ to satisfy Eq. (2). Furthermore, the velocity matrix $V(t)$ is initialized to zero.

The random values are generated by a high-performance NVIDIA CUDA Random Number Generation library called *cuRAND*, All dimensions are initialized independently. To avoid producing repetitious values in a different part of running the algorithm, the seed of the random generator has been set to the time of the system. In this kernel, $Q(t)$ and $V(t)$ is placed in the global memory and the number of blocks is the same as the size of the population ($Npop$), and the number of threads is set to the dimension n.

Observation step: Observation $\lll Npop, n \ggg (Q(t), SW(t))$: The observation process is done by Alg. 1 on GPU. This kernel gets $Q(t)$ as input and collapses the Q-bit states into a single state $x_i^1(t)$ with value 0, 1, 2, or 3 regarding α_i^d and β_i^d as probabilities. This kernel makes possible solution set $SW(t) = \{X_1(t), X_2(t), \ldots, X_{Npop}(t)\}$ where, $X_i(t) = \left(x_i^1(t), x_i^2(t), \ldots, x_i^n(t)\right)$. Like the initialization step, each agent is assigned to a block and each dimension is assigned to a thread. Therefore, the number of blocks and threads are $Npop$ and n respectively.

Fitness evaluation: The matrix $SW(t)$ is sent to the CPU, and the fitness values are computed by cores on the CPU using Eq. (9). Since computations of the fitness function involve clustering and using linked list structures, it seems to be more suitable to run on CPU cores by using Open-MP. The output of this function is $vectorfitness(t)$ which is sent to GPU.

Updating SB: Update_SB $\lll Npop, n \ggg (fitness(t), SW(t), SB(t))$: In this step, if the fitness of agent i at iteration t is better than its fitness value in iteration $t-1$, it will be replaced in SB. That is, SB saves the best value seen by each agent. At the first

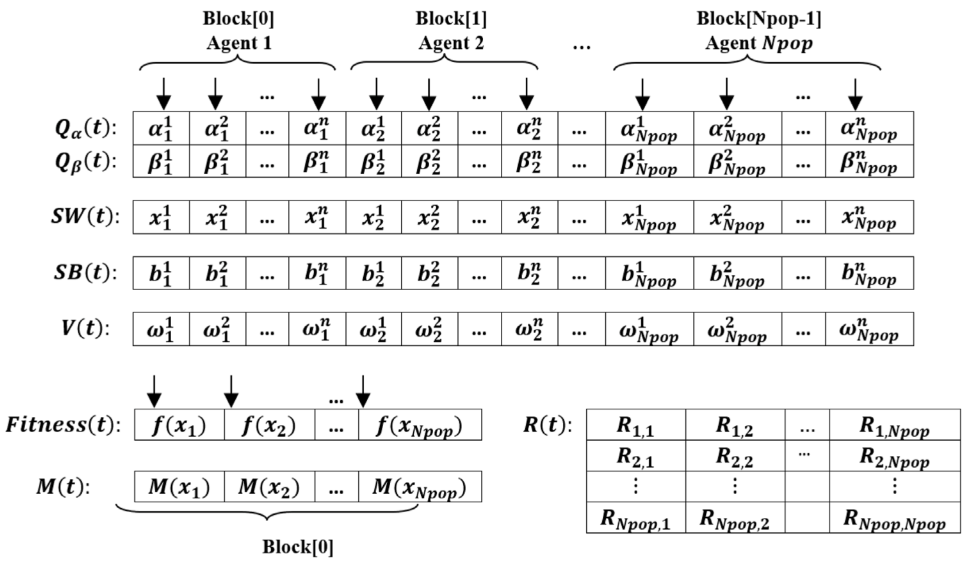

Fig. 3. The organization of the vectors in global memory.

iteration, $SW(0) = SB(0)$. This kernel reads $SW(t)$ and $fitness(t)$ from the global memory and writes $SB(t)$ on it. Similar to the last two kernels, the number of blocks and the block dimensions are equal to $Npop$ and n respectively.

Updating masses: Update_Mass $\lll 1, k \ggg$ $(Fitness(t), M(t))$: Regarding Eq. (3), updating masses involves computing the best, worst, and summation of the fitness values in iteration t. This kernel, which gets $Fitness(t)$ from the global memory and writes $M(t)$ on it, is called by a single block containing n threads to perform the reduction. The following relation can be used to calculate the mass of the agent i, which is equal to Eq. (3) [10].

$$M_i(t) = \frac{fit_i(t) - worst(t)}{\sum_{j=1}^{Npop}(fit_j(t) - Npop \times worst(t))} \tag{10}$$

Updating distances: Update_distances $\lll Npop \times Npop, n \ggg$ $(SB(t), R(t))$: This kernel is used by $Npop \times Npop$ blocks with n threads, to calculate the agents' distance matrix by Eq. (5). Each block updates $R_{ij(t)}$ that i and j correspond $blockId.x$ and $blockId.y$ respectively. Since $R_{ij}(t) = R_{ji}(t)$, for blocks with $blockId.x \le blockId.y$, the kernel copies $R_{ji}(t)$ while updating $R_{ij}(t)$.

Updating angular velocity: Update_Velosity $\lll Npop, n \ggg$ $(SW(t), SB(t), R(t), M(t), G(t), V(t))$: The kernel is called to calculate acceleration to update angular velocity using Eqs. (4) and (5). It reads $SW(t)$, $SB(t)$, $M(t)$, $R(t)$, and $G(t)$ from the global memory and computes the acceleration by Eq. (5), then updates velocity matrix $V(t)$ using achieved acceleration regarding Eq. (4). The number of blocks and the number of threads are $Npop$ and n respectively.

Updating Q-bits: Update_Qbits $\lll Npop, n \ggg$ $(Q(t), V(t), SB(t), SW(t), Q(t+1))$: This kernel update Q-bits using rotational angle as Eqs. (7) and (8). The inputs of this kernel are $Q(t), V(t), SB(t), SW(t)$ to calculate and update $Q(t+1)$. Like the last kernel, the number of blocks and the block dimensions are $Npop$ and n, respectively.

Applying Not Q-Gate: QGate $\lll 1, n \ggg$ $(Q(t+1), p_N)$: In this step, the proposed *Not Q-Gate* with the probability p_N is applied. In this kernel, an agent of $Q(t)$ is selected randomly with the probability p_N and change the amount of its α and β in all dimensions to promote more diversity to the population. This kernel uses one block with n threads.

This procedure is repeated until the termination criterion such as the maximum number of iterations (equal to ITE) is met. Not only parallelism can accelerate the algorithm, but also it can enhance the quality of results by increasing the size of the population. Besides, it helps to use the algorithm for a problem with a larger search space. Since the running time of the algorithm is declined by parallelism, the algorithm can be executed on more extensive networks at a reasonable time. Therefore, the parallel solving of the WSN problem can lead to scalability.

5. Experimental results

To evaluate the effectiveness and the speed of the proposed method, several tests are conducted on a WSN consisting of 900 sensors deployed in a grid-shaped area in the size of 30 × 30 square units. The goal is to assign the operational state of each sensor to optimize the energy consumption regarding the connectivity and specific-application considerations. The experiments performed on a computer with Intel©Core(TM) i7 CPU 1.80 GHz, 8.00 GB RAM and operating system Windows 10, 64-bit.

Table 1
The weighting coefficients related to the fitness function [1,2,6,7].

w_1	w_2	w_3	w_4	w_5	w_6	w_7
10^2	10^4	10^6	10^5	10	10^{-2}	10^{-2}

Table 2
The mean fitness value of IQQIGSA by two parameter setting: trial and error, and the Taguchi method for different size of the population.

Method	IQQIGSA (N_{pop} = 20)	IQQIGSA (N_{pop} = 50)	IQQIGSA (N_{pop} = 100)	IQQIGSA (N_{pop} = 200)
Trial and error	0.0209	0.021	0.0243	0.028
Taguchi method	0.0213	0.0218	0.0252	0.0288

Table 3
Parameter adjustment of the comparative algorithms [7].

Algorithm	Parameter value
BGA [1]	$p_m = 0.005, p_c = 0.8$
BPSO [2]	$w = 1, c_1, c_2 = 2$
Improved BQIGSA [6]	$G = (1 - 0.95 \times \frac{t}{ITE}) \times G_0$ and $G_0 = 3.78$, kbest is linearly relation decreasing from N_{pop} to 1
QQIGSA [7]	$G = (1 - 0.95 \times \frac{t}{ITE}) \times G_0$ and $G_0 = 0.125$, kbest is linearly function decreasing from N_{pop} to 1.
The Suggested IQQIGSA by Taguchi method	$G_0 = 0.2, \alpha = 0.95, p_N = 0.07$ kbest is linearly function decreasing from N_{pop} to 1.

5.1. Parameter settings

5.1.1. The related parameters to the fitness function

In the evolution of the fitness function and applying metaheuristic methods, there are several parameters to be adjusted that are described in this section. The weight coefficients notated by w_1, w_2, …, and w_7, used in Eq. (9) show the importance of each parameter in the fitness function. Table 1 presents the amount of each coefficient. The other parameters of the fitness function are set as follows. In SCE, the maximum number of active sensors that can communicate with CH is 15. ρ_d is set to 0.2 in SDE. In the SORE parameter, the signal range of LSR and HSR are considered 5 and 10 units, respectively. In OE, the proportion $x : y : z$ is considered as $20 : 2 : 1$ showing the energy consumption rate. In the CE parameter, μ is considered 1, and k is set to 3. In BCP, $PF_i^{[t]}$ is computed by the proportion $20 : 2 : 1$ for the operational modes of CH, HSR, and LSR. Besides, BRR (the battery reduction rates) of inactive, LSR, HSR, and CH modes are considered as 0, 0.01, 0.02 and 0.2, respectively.

5.1.2. The related parameters to the algorithms

The parameter setting of metaheuristics has an important role in the performance of the algorithm. Examining all possible parameter values is so time-consuming and complex. Therefore, for the proposed IQQIGSA we use the Taguchi method [24] for the design of the experiment (DOE) to find the best configuration of parameters. This method can decrease the required time to investigate the effects of multi parameters on performance to find the best one [18,25]. In this method, we firstly determine the effective parameters and their level. Then, we design the orthogonal array and execute the experiments and choose the best parameter configuration, based on the achieved results.

The main parameters of the proposed IQQIGSA are p_N as the probability of selecting an agent to be changed by *Not Q-Gate*, and parameters G_0 and α in relation $G = (1 - \alpha \times \frac{t}{ITE}) \times G_0$ used in Eq. (5). For each parameter, four levels of interest are determined in our DOE. The parameters and their levels are chosen as follows: four levels for $G_0 \in \{0.1, 0.2, 0.3, 0.4\}$; four levels for $\alpha \in \{0.8, 0.85, 0.9, 0.95\}$; and four levels for $p_N = \{0.02, 0.05, 0.07, 0.09\}$. The full parameter exploration for all possible combinations requires $4^3 = 64$ experiments, while the Taguchi method, using the orthogonal arrays, significantly reduces the number of executions. An orthogonal array $L_16(4^3)$ involves only 16 experiments is applied. Meanwhile, all the experiments with various parameter combinations are repeated 10 times and the best combination of parameters is reported base on the best-averaged fitness value. Regarding the experimental results among 16 examined combination by Taguchi's method combination ($G_0 = 0.2, \alpha = 0.95, p_N = 0.07$) performs better than the others to have the greatest averaged fitness value. Table 2 compare the averaged fitness value of IQQIGSA using parameter setting by trial and error (by fixing $G_0 = 0.125$ as QQIGSA [6] and gradual changing p_N) and the Taguchi method for different size of the population as $Npop = 20$, 50, 100, and 200. The trial and error finds the most appropriate combination as $G_0 = 0.125, \alpha = 0.95, p_N = 0.05$, and the Taguchi method suggests $G_0 = 0.2, \alpha = 0.95, p_N = 0.07$. Table 2 shows that the quality of results can improve by justifying parameters using the Taguchi design method.

The related parameters of the BGA, BPSO, QQIGSA, improved BQIGSA, and the suggested IQQIGSA are set as shown in Table 3, referring to the best values reported in the previous studies [1,2,5,6]. In all used algorithms the maximum number of iterations is set to 100.

Table 4
Comparison of the suggested IQQIGSA with the state-of-the-art ones during 100 iterations of the algorithms. The results are averaged on ten executions of the algorithms.

Parameter	BGA	BPSO	QQIGSA	Improved BQIGSA	IQQIGSA $N_{pop} = 20$	IQQIGSA $N_{pop} = 50$	IQQIGSA $N_{pop} = 100$	IQQIGSA $N_{pop} = 200$
Max fitness	0.135	0.0158	0.017	0.0207	0.0215	0.0221	0.0254	0.0291
Mean fitness	0.0124	0.0152	0.0162	0.0202	0.0213	0.0218	0.0252	0.0288
OE	5.0397	4.597	3.6017	3.0562	5.202	5.116	4.87	2.81
MRD	0.0166	0.0192	0.0215	0.0147	0.019	0.0165	0.021	0.014
CE	1259.26	1363.43	2985.45	2252.93	1158.51	1131.87	1186.93	2245.33
BCP	1064.1	1099.1	1194.53	554.86	1050.32	1028.71	971.22	519.602
SCE	0	0	0	0	0	0	0	0
SORE	0	0	0	0	0	0	0	0
SDE	0	0	0	0	0	0	0	0
CH	192	171.6	113.8	105.6	207.4	203.1	182.2	101.7
HSR	231.2	235.4	316.4	233.6	175.5	167.8	188.5	147.9
LSR	233.4	234.6	332.8	231.4	246.8	249.3	246.1	229.5

5.2. Comparative study

In this section, the performance of the suggested parallel IQQIGSA is studied in comparison with the adopted BGA [1], BPSO [2], QQIGSA [6] and improved BQIGSA [5] on WSN explained before. The near-optimal operational state of sensors is determined to maximize the fitness value formulated by Eq. (9). The average results of ten independent runs of the algorithm through 100 iterations are reported. Table 4 compares the algorithms based on the maximal and minimal value of the fitness as well as parameters like operational energy (OE), Mean Relative Deviation (MRD), Battery Capacity Penalty (BCP), Communication Energy (CE), Sensors-per-Cluster-head Error (SCE), Spatial Density Error (SDE), Sensor-Out-of-Range Error (SORE), and the average number of active sensors in operational modes of CH, HSR, and LSR.

Maximization of the fitness and minimization of the mentioned parameters are desired. It is observable that the suggested IQQIGSA with a population size of 20 has better results than previous methods including BGA [1], BPSO [2], QQIGSA [6] and improved BQIGSA [5]. By increasing the size of the population as $Npop = 50$, 100, 200, it is observable that the quality of the produced solutions by the proposed method has been grown. The last column of Table 4 is related to the suggested IQQIGSA with 200 agents. This column shows the best results compared to the others in terms of all the mentioned parameters except the communication energy (CE). This is for the trade-off of the operational energy (OE) and CE. The suggested algorithm decreases OE by reducing the number of CHs (cluster heads) that itself increases the CE parameter.

Fig. 4 compares the average progress of the fitness value of BGA, BPSO, QQIGSA, improved BQIGSA, and the suggested IQQIGSA during 100 iterations. It can be seen that the proposed IQQIGSA has the best progress compared to other algorithms. Furthermore, by increasing the population size (NP) the quality of solutions has been improved.

After using the proposed algorithm on WSN, the battery capacities are updated. Then, the dynamic optimal design algorithms (DODA) are executed on the WSN to study the effect of the used algorithms on the lifetime of the network. Fig. 5 illustrates the progress of the fitness for BGA, BPSO, Improved BQIGSA, QQIGSA and the suggested IQQIGSA along with 15 measuring cycles.

In WSN design, the measuring cycle is defined as a period for data collection with a specified operational mode. At the next measuring cycle, the operational mode of each sensor can be changed by the algorithm. To draw this figure, each algorithm runs repeatedly during 15 measuring cycles, considering the updated energy level of sensors at the end of each measuring cycle. Undoubtedly, the fitness value for all algorithms is decreased by time but the fitness value of the proposed method is at the top of the others. It shows that the proposed method can keep the network alive for more measuring cycles and the proposed IQQIGSA can increase the lifespan of the network better than others. The suggested IQQIGSA is the best method to keep the network alive and increase the lifetime of the network.

Increasing the number of agents may lead to enhancing the quality of solutions, but it increases the run time of the algorithm. Parallelism techniques can be used to manage this problem. The speedup is defined as Eq. (11) and parallel efficiency as Eq. (12), which are important measures to evaluate the quality of pluralization. In these equations, $T(np)$ is the execution time to run the algorithm on np processors.

$$S(np) = \frac{T(1)}{T(np)} \tag{11}$$

$$E(np) = \frac{S(np)}{np} \tag{12}$$

Tables 5 and 6 respectively report the speedup and efficiency of the parallel IQQIGSA using Open-MP (presented in Alg. 2) for a different number of cores varying from 2 to 8 ones. The effectiveness of the parallel implementation of the suggested algorithm by Open-MP in the WSN problem is observable in these tables. Regarding Table 5, the speedup averagely enhances from 1.83 to 4.24 by varying the number of threads from 2 to 8 ones. The reported speedup and efficiency values are evaluated based on the average of ten runs.

The reported results of these tables show that by increasing the number of CPU cores from 1 to 8, the speedup is increased and the efficiency is decreased. Although there is no significant difference among the results of columns for different sizes of populations.

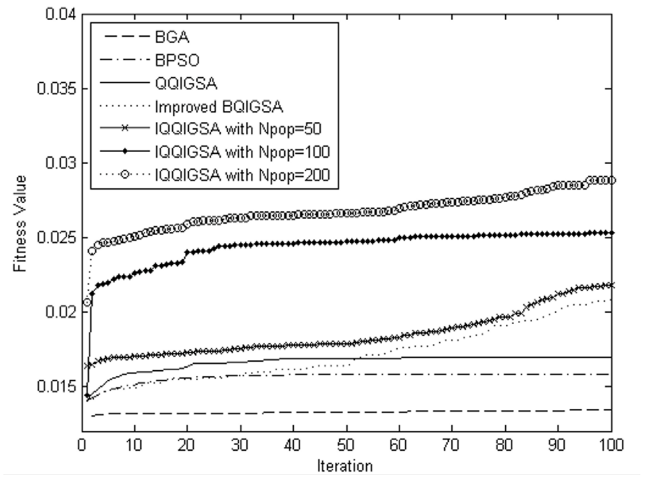

Fig. 4. The performance evaluation of the proposed IQQIGSA in comparison with other methods along 100 iterations in terms of the fitness values. The results are averaged over ten executions.

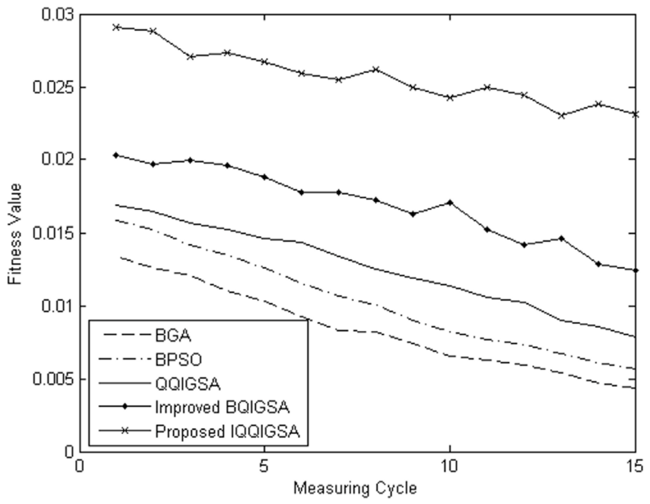

Fig. 5. The Performance evaluation of the DODA by BGA, BPSO, QQIGSA, improved BQIGSA, and the proposed IQQIGSA during 15 measuring cycles in terms of fitness values.

Table 5
Speedup of the suggested parallel IQQIGSA using Open-MP by changing the size of the population and the number of cores.

Num of cores	2	3	4	5	6	7	8
$N_{pop} = 20$	1.84	2.41	3.06	3.33	3.46	3.92	3.97
$N_{pop} = 50$	1.80	2.45	3.11	3.24	3.68	4.04	4.35
$N_{pop} = 100$	1.84	2.35	2.71	2.89	3.60	4.01	4.28
$N_{pop} = 200$	1.86	2.44	3.14	3.36	3.60	4.04	4.38

Table 7 shows the results of applying the proposed CPU–GPU heterogeneous paralleling technique on IQQIGSA for the optimal design of WSN which is presented in Alg. 3 in comparison with using just a multi-core CPU to compute the fitness values. The results show the fitness function of the algorithm in Alg. 2 is the most time-consuming section of the algorithm. Parallel computing by Open-MP implemented of the algorithm is averagely 4.24 times faster than the serial version on an 8-core CPU. The CPU–GPU heterogeneous implementation in which the computation of the fitness function is done by CPU and the other sections are executed on GPU, improves the speedup about 8.06 times more than the sequential method and about 2 times faster than the first algorithm. The results indicate that by increasing the population size the speedup has been improved using the GPU.

Comparison of the speedup for parallel IQQIGSA on WSN by Multi-Core and the CPU–GPU heterogeneous implementation

Table 6
The efficiency of the suggested parallel IQQIGSA using Open-MP by changing the size of the population and the number of cores.

Num of cores	2	3	4	5	6	7	8
$N_{pop} = 20$	0.92	0.80	0.76	0.67	0.58	0.56	0.50
$N_{pop} = 50$	0.9	0.82	0.78	0.65	0.61	0.58	0.54
$N_{pop} = 100$	0.92	0.78	0.68	0.58	0.60	0.57	0.53
$N_{pop} = 200$	0.93	0.81	0.79	0.67	0.60	0.57	0.54

Table 7
Comparison of the speedup for parallel IQQIGSA on WSN by Multi-Core and the CPU–GPU heterogeneous implementation.

Population size	Multi-Core CPU (Alg. 2)	CPU–GPU Heterogeneous (Alg. 3)
$N_{pop} = 20$	3.97	7.52
$N_{pop} = 50$	4.35	8.43
$N_{pop} = 100$	4.28	9.02
$N_{pop} = 200$	4.38	9.45

The results show that not only the parallel implementation of the proposed IQQIGSA help to decrease the execution time of the algorithm, but also it can help to enhance the quality by allowing the increment in the number of agents. Furthermore, the parallel implementation of multi-core architecture can help to decrease the power consumption in the mentioned WSN. Because this is the nature of multicore platforms where several low-power cores are on the chip. This architecture which is using parallelism by multiple cores can enhance the throughput without increasing the power of the processor. Besides, parallelism allows the methods to be used on more extensive networks which compose more sensors and run in a reasonable time.

6. Conclusion

In this study, a modified version of Quadri-valent Quantum-Inspired Gravitational Search Algorithm (QQIGSA) was introduced by a Not Quantum-Gate (*Not Q-Gate*). To enhance the speed of this algorithm, we used parallel platforms including a multi-core platform and a heterogeneous one using the central processing unit and graphical processing unit called Central Processing Unit-Graphic Processing Unit (CPU–GPU) platform. The parallel improved QQIGSA was adopted on a wireless sensor network to specify the operational mode of sensors, as the connectivity, specific application parameters and energy-related parameters are minimized. The sensor of this network can operate in one of the four possible operational states including low signal range, high signal range, and cluster head and inactive. This is the reason that a Quadri-valent algorithm has been developed.

The results report a satisfactory performance of the suggested method based on the quality of the solutions in comparison with the state-of-the-art ones. Moreover, parallelism based on using a multi-core platform enhanced the speedup over 4 times. Besides, the speedup was more than 8 times, when the CPU–GPU heterogeneous platform was employed. Paralleling this algorithm on the optimal design of wireless sensor Networks is helpful considering four aspects. The first one is decreasing the execution time of the code. The second one is the possibility of applying the algorithm in more extensive wireless sensor Networks. The third one is increasing the quality of results by allowing an increase in the number of agents, and the last one is effective usage of the base station power by applying a multi-core processor.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] Ferentinos KP, Tsiligiridis TA. Adaptive design optimization of wireless sensor networks using genetic algorithms. Comput Netw 2007;51(4):1031–51.
[2] Hojjatoleslami S, Aghazarian V, Aliabadi A. DE based node placement optimization for wireless sensor networks. In: 2011 3rd International Workshop on Intelligent Systems and Applications. IEEE; 2011, p. 1–4.
[3] Sammut C, Webb GI. Encyclopedia of machine learning and data mining. Springer; 2017.
[4] Mittal S, Vetter JS. A survey of CPU-GPU heterogeneous computing techniques. ACM Comput Surv 2015;47(4):1–35.
[5] Mirhosseini M, Barani F, Nezamabadi-pour H. Design optimization of wireless sensor networks in precision agriculture using improved BQIGSA. Sustain Comput Inform Syst 2017;16:38–47.
[6] Mirhosseini M, Barani F, Nezamabadi-pour H. QQIGSA: A quadrivalent quantum-inspired GSA and its application in optimal adaptive design of wireless sensor networks. J Netw Comput Appl 2017;78:231–41.
[7] Mirhosseini M, Fazlali M, Gaydadjiev G. A parallel and improved quadrivalent quantum-inspired gravitational search algorithm in optimal design of WSNs. In: International Congress on High-Performance Computing and Big Data Analysis. Springer; 2019, p. 352–66.
[8] Crainic TG. Parallel meta-heuristic search. CIRRELT; 2015.
[9] Mahmoud KR, Hamad S. Parallel implementation of hybrid GSA-NM algorithm for adaptive beam-forming applications. Prog Electromagn Res 2014;58:47–57.

[10] Zarrabi A, Samsudin K, Karuppiah EK. Gravitational search algorithm using CUDA: a case study in high-performance metaheuristics. J Supercomput 2015;71(4):1277–96.
[11] Rashedi E, Nezamabadi-Pour H, Saryazdi S. GSA: a gravitational search algorithm. Inform Sci 2009;179(13):2232–48.
[12] Nezamabadi-pour H. A quantum-inspired gravitational search algorithm for binary encoded optimization problems. Eng Appl Artif Intell 2015;40:62–75.
[13] Ibrahim AA, Mohamed A, Shareef H. A novel quantum-inspired binary gravitational search algorithm in obtaining optimal power quality monitor placement. J Appl Sci 2012;12(9):822–30.
[14] Wang Y, Gao S, Yu Y, Wang Z, Cheng J, Yuki T. A gravitational search algorithm with chaotic neural oscillators. IEEE Access 2020;8:25938–48.
[15] Lei Z, Gao S, Gupta S, Cheng J, Yang G. An aggregative learning gravitational search algorithm with self-adaptive gravitational constants. Expert Syst Appl 2020;152:113396.
[16] Wang Y, Yu Y, Gao S, Pan H, Yang G. A hierarchical gravitational search algorithm with an effective gravitational constant. Swarm Evol Comput 2019;46:118–39.
[17] Gao S, Vairappan C, Wang Y, Cao Q, Tang Z. Gravitational search algorithm combined with chaos for unconstrained numerical optimization. Appl Math Comput 2014;231:48–62.
[18] Ji J, Gao S, Wang S, Tang Y, Yu H, Todo Y. Self-adaptive gravitational search algorithm with a modified chaotic local search. IEEE Access 2017;5:17881–95.
[19] Rashedi E, Rashedi E, Nezamabadi-pour H. A comprehensive survey on gravitational search algorithm. Swarm Evol Comput 2018;41:141–58.
[20] Candan C, Dréo J, Savéant P, Vidal V. Parallel divide-and-evolve: Experiments with OpenMP on a multicore machine. In: Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation, 2011: p. 1571–1578.
[21] Moradi E, Fazlali M, Malazi HT. Fast parallel community detection algorithm based on modularity. In: 2015 18th CSI International Symposium on Computer Architecture and Digital Systems (CADS). IEEE; 2015, p. 1–4.
[22] Fazlali M, Zakerolhosseini A, Sabeghi M, Bertels K, Gaydadjiev G. Data path configuration time reduction for run-time reconfigurable systems. In: ERSA. 2009, p. 323–7.
[23] Mirhosseini M, Fazlali M. Parallel and exact method for solving n-similarity problem. J Electr Comput Eng Innov (JECEI) 2020;8(2):193–200.
[24] Altland HW. Computer-Based Robust Engineering: Essentials for DFSS. Taylor & Francis; 2006.
[25] Chui KT, Liu RW, Zhao M, De Pablos PO. Predicting students' performance with school and family tutoring using generative adversarial network-based deep support vector machine. IEEE Access 2020;8:86745–52.

Mina Mirhosseini received M.Sc. in Computer Science from Shahid Bahonar University of Kerman. She is a lecturer at the department of science, Higher Education Complex of Bam. Currently, she is working toward her Ph.D. in computer science at Shahid Beheshti University. Her research interests include parallel processing, metaheuristic, text processing and wireless sensor networks.

Mahmood Fazlali received M.Sc. from University of Isfahan in 2004, and Ph.D. from Shahid Beheshti University (SBU) in 2010 both in computer architecture. He performed researches on reconfigurable computing systems at Technical University of Delft (TUDelft) as a postdoc researcher. Now, he is working as assistant professor at SBU. His research interest includes parallel processing, data science and reconfigurable computing.

Hadi Tabatabaee Malazi received his Ph.D. in computer engineering from the University of Isfahan (2012). He was an assistant professor at Shahid Beheshti University (2013–2019). He is a research fellow at the School of Computer Science and Statistics (Trinity College Dublin) and a member of Connect Research Centre. His area of research is edge computing in smart city applications.

Sayyed Kamyar Izadi obtained M.Sc and Ph.D. in software engineering from Iran University of Science and Technology (IUST) in 2003 and 2011 respectively. He joined research group lead by Prof. Härder at Technical University of Kaiserslautern participated in the XTC project in 2008. Now he is assistant professor at Alzahra University. His research interest includes Relational and non-relational Database Systems.

Hossein Nezamabadi-pour received B.Sc. in Electrical Engineering from Shahid Bahonar University of Kerman in 1998, then M.Sc. and Ph.D. in Electrical Engineering from Tarbait Moderres University in 2000 and 2004, respectively. He is working as full professor at Electrical Engineering at Shahid Bahonar University of Kerman, Iran. His research interests include image processing, pattern recognition, soft computing, and evolutionary computation.