sciendo

JAISCR, 2019, Vol. 9, No. 3, pp. 205 – 218
10.2478/jaiscr-2019-0004

SOLUTION OF LINEAR AND NON-LINEAR BOUNDARY VALUE PROBLEMS USING POPULATION-DISTRIBUTED PARALLEL DIFFERENTIAL EVOLUTION

Amnah Nasim[1], Laura Burattini[1,*], Muhammad Faisal Fateh[2], and Aneela Zameer[2]

[1]*Department of Information Engineering (DII), Università Politecnica delle Marche, Via Brecce Bianche, 60131 Ancona, Italy*

[2]*Department of Computer and Information Sciences (DCIS), Pakistan Institute of Engineering and Applied Sciences, Nilore, 44000 Islamabad, Pakistan*

[*]*E-mail: l.burattini@univpm.it*

Submitted: 18th October 2018; Accepted: 20th January 2019

Abstract

Cases where the derivative of a boundary value problem does not exist or is constantly changing, traditional derivative can easily get stuck in the local optima or does not factually represent a constantly changing solution. Hence the need for evolutionary algorithms becomes evident. However, evolutionary algorithms are compute-intensive since they scan the entire solution space for an optimal solution. Larger populations and smaller step sizes allow for improved quality solution but results in an increase in the complexity of the optimization process. In this research a population-distributed implementation for differential evolution algorithm is presented for solving systems of 2^{nd}-order, 2-point boundary value problems (BVPs). In this technique, the system is formulated as an optimization problem by the direct minimization of the overall individual residual error subject to the given constraint boundary conditions and is then solved using differential evolution in the sense that each of the derivatives is replaced by an appropriate difference quotient approximation. Four benchmark BVPs are solved using the proposed parallel framework for differential evolution to observe the speedup in the execution time. Meanwhile, the statistical analysis is provided to discover the effect of parametric changes such as an increase in population individuals and nodes representing features on the quality and behavior of the solutions found by differential evolution. The numerical results demonstrate that the algorithm is quite accurate and efficient for solving 2^{nd}-order, 2-point BVPs.

Keywords: parallel evolutionary algorithms, differential evolution, boundary value problems, optimization

1 Introduction

Several evolutionary algorithms have been presented in recent years because of the huge success achieved in finding an optimum solution to problems for which traditional derivative-based methods fail. In the case where a derivative of the problem does not exist or is constantly changing, the traditional derivative solution can easily get stuck in the local optima or does not factually represent a constantly changing solution. To find a quality solution for such problems, derivative-free techniques are used such as genetic algorithms (GAs), genetic programming (GP) etc. Genetic or Evolutionary algorithms use a metaheuristic approach to find an optimum solution to a given problem by

generating a large population of individuals in the possible solution space. However, due to an increase in complexity and number of evaluations of the objective function required to reach a termination criterion, its application to complex problems or larger search spaces result in excessive algorithm runtimes. To solve this problem, distributed evolutionary algorithms (dEAs) [1, 2] were introduced to speed up simple evolution-based computation strategies. Differential evolution (DE) provides such a genuine heuristic and is used for global optimization in a variety of real-world problems in both discrete and continuous domains [3, 4]. Differential evolution is a multi-point, derivative-free approach of scanning a solution space for global optimization [5] introduced by Price and Storn in 1995. Like all other evolutionary algorithms (EAs), DE performs population-based optimization targeting the starting point problem by sampling the fitness function at randomly chosen multiple initial points. However, DE is compute-intensive since it scans the entire solution space for an optimal solution. The fact that in classic DE algorithm and its proposed variants, each individual in the population is not compared against all the individuals in the current population, but only against its counterpart in the current population which replaces if better fitted, is a very important characteristic concerning the parallelization of the DE algorithm. This feature has been exploited and successfully applied in numerous real-world application areas such as large scale multi-criteria and multi-population optimization problems [6, 7, 8], digital filter design [3], industrial system design [9, 10], computational systems biology [11], gene regulatory networks [12, 13], identification of experimental data [14], transmission systems [15], designing task scheduler for parallel operating systems [16] etc. Numerous distributed architectures have been proposed for specified real-world applications to cater for this computation intensive problem. In terms of Flynn's taxonomy [17], the parallelization of an evolutionary algorithm can be made at one of the following levels: objective function evaluation level (master-slave model), population level (multi-population model, called also an island model or migration model) and elements level (cellular model). The cellular model leads to fine-grained parallelization while the former two models lead to coarse-grained parallelization. Two main architectures [18] considered advantageous for parallel computation of the Differential Evolution algorithm are first a population distribution strategy which results in a significant reduction in processing time but with near-optimum solution accuracy. Second, an island topology, that provides a significant improvement in finding the optimum solution as the search space is increased due to the introduction of more islands and each island is configured with a different initial seed. But the ring topology was unable to provide any significant improvement in the computation time. Hence, both architectures suffer a trade-off in accuracy in finding an optimal solution and the computation time.

In 2006, Ntipteni et al. [19] developed an asynchronous parallel Differential Evolution framework for a cluster of computers in Windows environment using master-slave architecture. The information of current population is stored in a common folder. Every population individual evolves on a separate machine independent from the rest of the population and updates the information stored in the common global variable. The procedure is tested in two airfoil optimization problems and the parallel code is compared to a serial one, with respect to the convergence behavior, the quality of the optimum solution and the total computation time. Zaharie et al. [6] proposed a coarse-grained parallel architecture for an adaptive differential evolution algorithm based on the multi-population random connection topology. The results show a significant speedup for parallel execution on the cluster and the global optimum is found with a higher probability even if more than one local optimum exist in the solution space. Recently, Fateh et al. [20] presented a nature-inspired framework for solving second order two-point boundary value problems (BVPs) using Differential Evolution (DE). They evaluated the method using five benchmark boundary value problems in linear and nonlinear regime of BVPs and reported a marked improvement in the quality and precision of their solution. The authors conclusively suggested that a reduction in error can be further achieved by increasing the number of nodes and/or the accuracy of derivative approximation techniques. While smaller step size yields more accurate answers, it entails higher computational cost.

Classical Differential Evolution performs random search for an ideal candidate by scanning the entire solution space populated by competing candidates. A fitness value is calculated for every individual in the population to give a quantitative measure of how accurately the candidate solves the problem. Like other evolutionary algorithms, DE method includes different steps as: population initiations, trial-vector generation, crossover, new fitness evaluation and selection of the comparatively fitter candidate. Preliminary tests indicate that the fitness calculation for every population individual once per generation takes up 80% of the total time. Hence, in this work a multi-core implementation based on master-slave [18] configuration for Differential Evolution algorithm is presented and tested for selected 2^{nd} order 2-point BVPs. Population-distributed models include master-slave, island, cellular, hierarchical, and pool models, which parallelize an optimization task at population, individual, or operation levels. An increase in the size of the population or features impairs the performance of boundary value problems (BVPs) in terms of application to real-world scenarios. To solve this problem a population-based [1, 21] parallelization scheme has been implemented using a fine-grained master-slave architecture as shown in Figure 2. The algorithm defines population as a single global data storage variable. The fitness of individuals and the genetic operations are carried out by the slave machines in parallel. Each machine updates one population individual in any two consecutive generations. The features of the proposed framework are: Scalability for complex system modeling i.e. higher number of nodes (small step-size), ability to search highly accurate solutions in reduced time and adaptability to different objective functions for different benchmark BVPs. Population-distributed high-performance implementation of the proposed algorithm is done using MATLAB® parallel computing toolbox. A performance evaluation is conducted on 4 selected benchmark BVPs to show the solution accuracy of the proposed method. Finally, the runtime for sequential program is compared with the multicore implementation on a standalone machine to assess computational efficiency.

The paper is organized as follows: Section 1 gives a summary of the previous work done to achieve computational efficiency. Section 2 describes the methodology and architectural framework used for the proposed PDDE algorithm. Section 3 presents achieved results and a discussion on the benefits realized through efficient implementation of Differential Evolution. Section 4 concludes the research work with some possible future extensions.

2 Population-Distributed Parallel Differential Evolution

Population-Distributed parallel Differential Evolution (PDDE) employs fine-grained parallelism in which each individual and competing mutated vector is assigned to a different processor and the fitter individual is correspondingly updated in the global population matrix by each processor. The result of each generation depends upon the population matrix gathered from the previous generation hence parallelism is rather achieved with distribution of population individuals on numerous processors than on generations.

The solution for BVPs is based on fitness function that approximates second-order derivatives and consequently, the error decrements follows $O(h^2)$ process. A vector population is generated such that the allowed parameter region is entirely covered. All vectors are indexed according to the number of generation because each of them must compete against the previous and maybe next generation too if selected. The vector subscripts in the following description represent first the number of generation to which the vectors belong and the row index in the population and trial matrix. The initial unevolved population generated is indexed **0** and the one after first evolution is indexed **1** so on until the final generation $\boldsymbol{n}$. Two vectors $\boldsymbol{u}_{01}$ and $\boldsymbol{v}_{01}$ are randomly selected from the initial vector population. $\boldsymbol{s}_{01}$ is a third randomly picked vector different than $\boldsymbol{u}_{01}$ and $\boldsymbol{v}_{01}$ which yields the trial vector $\boldsymbol{t}_{01}$ using the weighted sum and mutation factor ($\beta = 0.5$) and placed at row number **1** in the trial matrix. Similarly, $\boldsymbol{u}_{02}$, $\boldsymbol{v}_{02}$ and $\boldsymbol{s}_{02}$ are generated anew and $\boldsymbol{t}_{02}$ is the trial vector that constitutes row number **2** in the trial matrix until there is a separate mutated trial vector generated to compete with every population individual. Now the competing initial population matrix and mutated trial vector ma-

trix are distributed for the crossover, fitness calculation and replacement operation. The trial vector $\boldsymbol{t}_{01}$ competes with vector **1** of the population based on fitness value on processor **1**. $\boldsymbol{t}_{02}$ competes against vector no. **2** of the population on processor number **2**. Two operations can then occur at every processor: replacement or survival. Each processor updates the corresponding vector in the population matrix according to the fitness value comparison. Hence in the current hardware resource scenario, 8 processors make updates to the population matrix simultaneously. A new evolved population matrix enters the next generation and a corresponding trial matrix is generated again. This process terminates if the termination criteria are met (fitness, number of generations etc.). The method is described stepwise as follows:

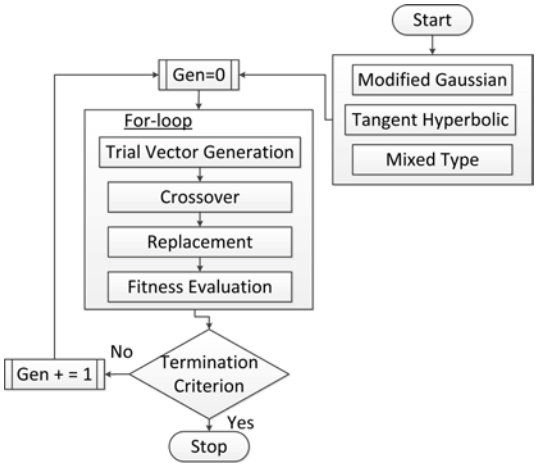

Figure 1. Serial Differential Evolution Algorithm

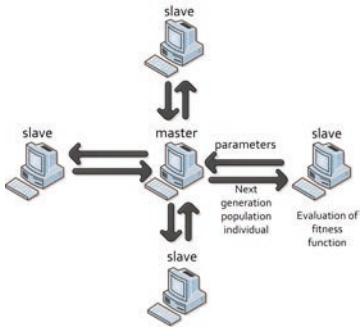

Figure 2. Master-Slave architecture

2.1 Population Initialization

Initial population is generated using the boundaries for the given possible solution space. Mixed-type population initiation scheme is chosen here for all benchmarks due to its competitive nature [20, 21]. A mixed-type population is defined as half of the number of individuals generated through tangent-hyperbolic function and rest half generated using modified Gaussian function. Using the given initial and final boundary values for all tested BVPs, the mixed-type population matrix is generated as

$$Pop(i,f) = a + 0.5\,(b-a)\,[1+\tanh\left((i-\mu)/\sigma\right)], \tag{1}$$

where,

$$i = 1 - \frac{N}{2}, j = 2-(N-2),$$

and

$$Pop(i,f) = a + \frac{b-a}{N+1}i + \exp\left(-(i-\mu)^2 \Big/ \left(2\sigma^2\right)\right), \tag{2}$$

where,

$$i = \left(\frac{N}{2}+1\right) - N, j = 2-(N-2).$$

2.2 Trial Matrix Generation

A matrix of trial vectors using current to best mutation strategy with scaling factor as $\beta = 0.5$ and each trial vector calculated as

$$t_{ij} = s_{ij} + \beta(u_{ij} + v_{ij}). \tag{3}$$

In 3 'i' is the number of current generation and 'j' is the row index for every vector t_{ij} in the trial matrix (total number of rows in the trial matrix equals the number of population individuals), s_{ij}, u_{ij} and v_{ji} are randomly picked vectors from the feasible solution space in any given generation.

2.3 Scatter Population

The multicore implementation uses 'parfor' directive from the MATLAB® Parallel Computing Toolbox to divide and assign one population individual and one trial vector to each core. Hence the distributed algorithm utilizes all cores and calculates the next generation population matrix by iterating over all individuals in the population.

2.4 Crossover

Exponential crossover is performed at this stage to produce offspring using individual population and trial vectors. Each core carries out the calculation of one offspring at a time. A high crossover

probability ($p = 0.99$) is used in this work since empirical study shows that it improves the convergence rate.

2.5 Fitness Calculation and Replacement

Each core processes one population individual calculates the fitness of the parent and offspring. If the parent vector gives the highest objective function value, it survives to the next generation. If the offspring produced after mutation and crossover has better fitness function value, it replaces the parent vector in the next generation.

2.6 Gather Population

Finally, every individual vector with fitness better than or same as its parent is gathered from all slave machines and sent to the master where this population array is then processed further for the next generation and a new trial matrix.

2.7 Termination Criteria

Two possible termination measures are used in this algorithm. The program terminates if the required fitness of 10^{-5} is achieved or if the maximum number of generations 500 is reached. In the current population-distributed differential evolution implementation, mutation and crossover occur at every step. For every new generation, a new trial matrix is generated. So, Mutation occurs after every generation instead of once in 10 or 15 generations. In all implementations, the initial population, and trial vectors are pre-calculated and stored in two different matrices.

Figure 1 shows a straightforward flow diagram for classical Differential Evolution with abovementioned functions being performed in a sequential manner for comparison. Figure 3 elucidates our parallel approach in which the input initial population is distributed to slave machines where most of the computation is carried out. The slave machines calculate fitness, perform crossover to create individual offspring, calculate the fitness of offspring and finally, update the best fitted population individual in the global population variable. Following the trend of sequential algorithm, each individual in parallel process competes with any other (thus mutation and crossover are global).

2.8 Selected 2^{nd} Order 2-point benchmark Boundary Value Problems

Following are the selected second-order, two-point boundary-value problems used to depict our results:

Example 1:
$\frac{d^2y}{dx^2} = 2\frac{dy}{dx} - y - 3, \quad 0 \le x \le 1$
Boundary Condition: y(0)=-3, y(1)=-2.264241
Exact Solution: $2xe^{(x-2)} - 3$ Source: [22]

Example 2:
$\frac{d^2y}{dx^2} = 4y, \quad 0 \le x \le 1$
Boundary Condition: y(0)=1, y(1)=0.1353
Exact Solution: e^{-2x} Source: [22]

Example 3:
$\frac{d^2y}{dx^2} = \frac{1}{8}(32 + 2x^3 - y\frac{dy}{dx}), \quad 1 \le x \le 2$
Boundary Condition: y(1)=17, y(2)=12
Exact Solution: $x^2 + \frac{16}{x}$ Source:[22]

Example 4:
$\frac{d^2y}{dx^2} = -(x+1)\frac{dy}{dx} + 2y + (1-x^2)e^{(}-x), 0 \le x \le 1$
Boundary Condition: y(0)=-1, y(1)=0
Exact Solution: $(x-1)e^{-x}$ Source:[23]

2.9 Experimental Analysis

Computation of results include a mixed-type population initiation scheme along with the exponential crossover with a high crossover probability of 0.99 for all four numerical problems. The number of node points tested are 11, 111 and 1111 for sizes of population varying from 1000, 4000, 6000 and 10,000 individuals.

Hence the number of parameter combinations for each of the four problems becomes 12 (3x4). The population size of 1000 and 11 nodes are least used for comparison of accuracy and achieved speed up with the results presented by Fateh et al. [20]. And the rest of the parameters are explored additionally to study the benefits of parallel architecture for large-scale data. The parameters used in the current PDDE algorithm are given in Table 1. The platform used for these calculations was an Intel® Optiplex 9010 with four cores running at 3.4 GHz each. All the results presented correspond to an average of 10 simulations for each example.

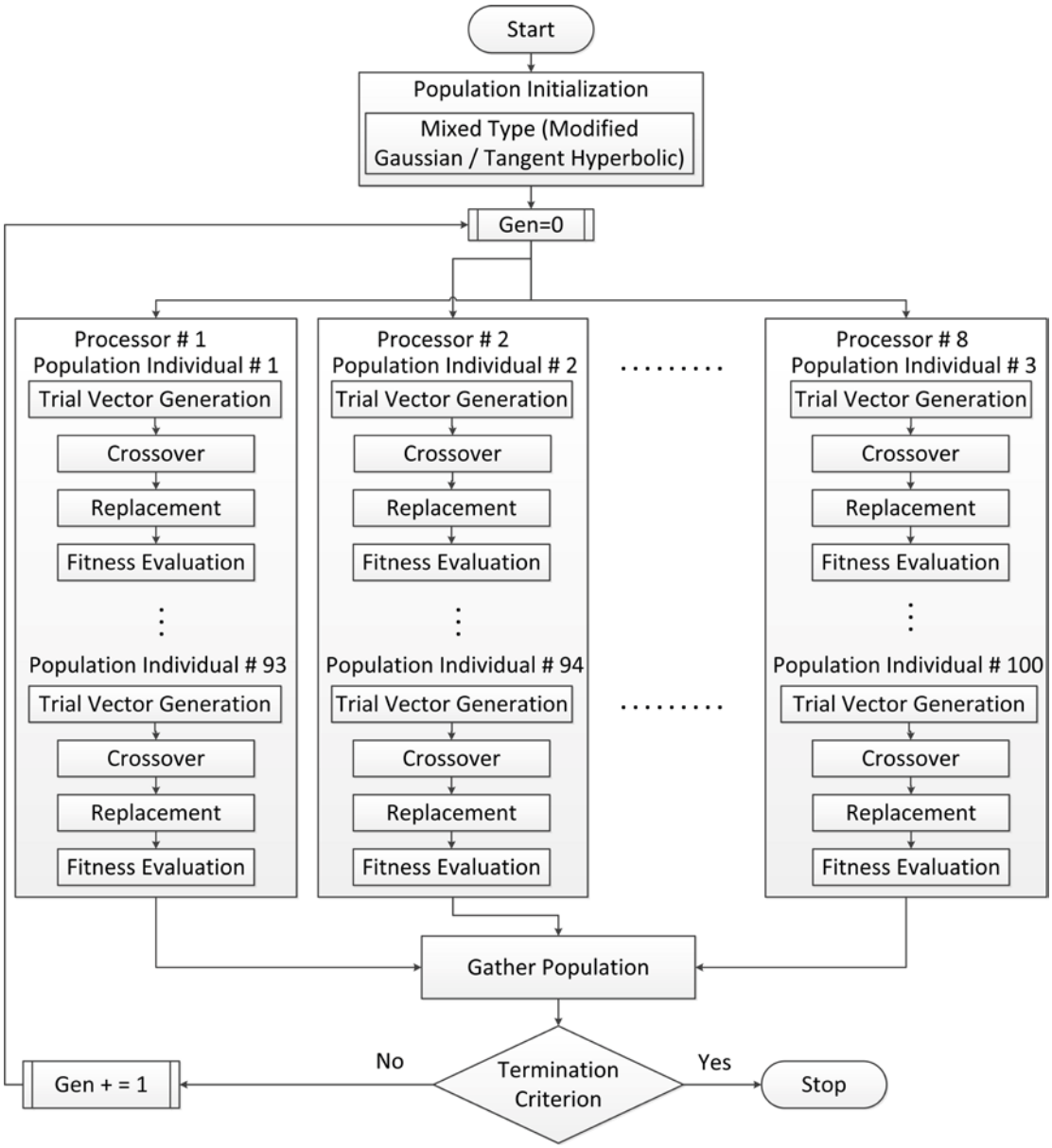

Figure 3. Framework for Population-Distributed parallel Differential Evolution

3 Results and Discussion

The results and discussion section are divided into two parts: 1) The residual error calculations show the accuracy of the proposed algorithm and 2) Speed of execution uses runtime and speedup achieved to show the performance of the parallelism for each of the four benchmark BVP cases mentioned before. A summary of parameters used in the current PDDE algorithm are given in Table 1.

Table 1. Parameter Values for PDDE

Parameters	Values
Initialization Scheme	Tan Hyperbolic
Crossover Probability	0.99
Mutation Scaling Factor	0.5
Mutation Strategy	Current to Best
Number of Nodes	11, 111, 1111
Number of Population Individuals	1000, 4000, 6000, 10,000
Maximum Number of Generations	500

3.1 Residual Error

Tables 2, 3, 4 and 5 show detailed nodal solution and error statistics of the problem cases solved using PDDE methodology.

The average residual error was found to be 2.7472×10^{-4} (PDDE 4) and 2.5474×10^{-4} (PDDE 8) for Example 1, 2.7900×10^{-4} (PDDE 4) and 2.7871×10^{-4} (PDDE 8) for Example 2, 6.4435×10^{-4} (PDDE 4) – 6.4380×10^{-4} (PDDE 8) for Example 3 and 1.7180×10^{-4} (PDDE 4) – 1.7082×10^{-4} (PDDE 8) for Example 4. This error comparison of the nodal points is also represented in Figure 4 as calculated by PDDE 4 and PDDE 8 in comparison to residual error obtained at the same points with sequential DE. Figure 5 represents the true node point values of the solution achieved with 11 nodes using both PDDE 4 and PDDE 8 in comparison to residual error obtained at the same points with sequential DE and exact solution for Example 1, 2, 3 and 4.

The accuracy of the solution as found by the proposed algorithm calculated at 4 and 8 cores using PDDE presented in terms of nodal values and nodal residual error in Figure 4 and Figure 5 respectively is comparable to sequential DE and exact solutions for all the selected boundary value problems. This shows that the accuracy of the proposed algorithm in finding an optimum for the tested BVPs is high within and the convergence is found in 95 – 110 generations.

3.2 Speed of execution

Keeping the residual nodal error intact as shown in Figure 5 for the exact, sequential DE and PDDE at 4 and 8 cores, the time required to achieve the required fitness value of 10^{-5} is decreased by PDDE.

Figure 6 shows the execution times achieved for population array computation for all four numerical problems using DE and PDDE on 2, 4, 6 and 8 cores. The average runtime over the 12 parameter combinations (population sizes: 1000, 4000, 6000, 10000 and nodes: 11, 111, 1111) was found to be 184.7287 (PDDE 4) and 129.4244 (PDDE 8) seconds for Example 1, 264.7667 (PDDE 4) and 172.5014 (PDDE 8) seconds for Example 2, 296.4331 (PDDE 4) and 182.5171 (PDDE 8) seconds for Example 3 and 362.5911 (PDDE 4) and 213.8212 (PDDE 8) seconds for Example 4. The results show that the runtime has decreased for all four numerical problems using PDDE on 4 and 8 cores. Highest runtime values are observed in Example 4, which is due to the complex nature of the objective value function of the said numerical problem.

Figure 7 shows the speedups achieved for population array computation for all four numerical problems using DE and PDDE on 2, 4, 6 and 8 cores. The average speedup over the 12 parameter combinations (population sizes: 1000, 4000, 6000, 10000 and nodes: 11, 111, 1111) was found to be 2.85× (PDDE 4) and 4.10× (PDDE 8) for Example 1, 2.36× (PDDE 4) and 3.60× (PDDE 8) for Example 2, 2.69× (PDDE 4) and 4.08× (PDDE 8) for Example 3 and 2.66× (PDDE 4) and 4.62× (PDDE 8) for Example 4. An overall average of 2.64× speedup is achieved for 4 cores and 4.10× speedup is achieved for 8 cores with the maximum of 6.08× for Example number 4 with population×nodes matrix 1000×111 and minimum of 1.25× for Example number 2 with population×nodes matrix 6000×111.

An increase in the runtime of the algorithm is observed with corresponding increase in the popu-

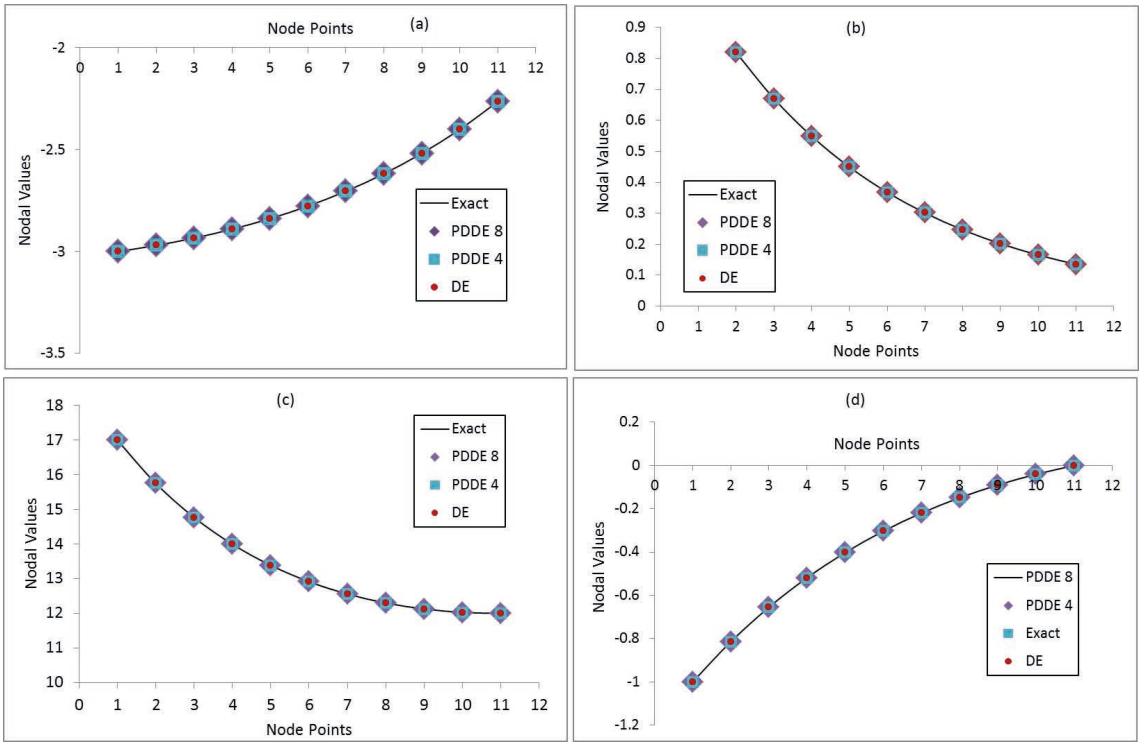

Figure 4. Nodal values computed by PDDE at 4 and 8 cores compared with Exact and DE reported solution plotted against node points for boundary-value problems, (a) Example 1 (b) Example 2 (c) Example 3 (d) Example 4

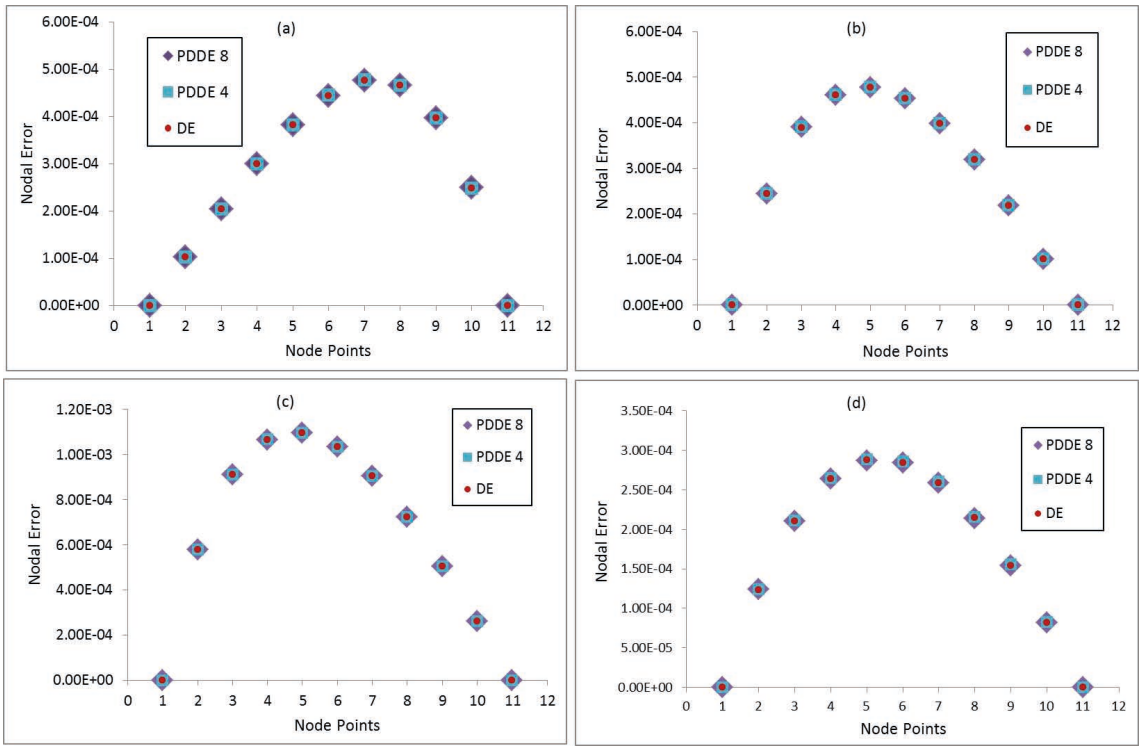

Figure 5. Nodal error computed by PDDE at 4 and 8 cores compared with Exact and DE reported solution plotted against node points for boundary-value problems, (a) Example 1 (b) Example 2 (c) Example 3 (d) Example 4

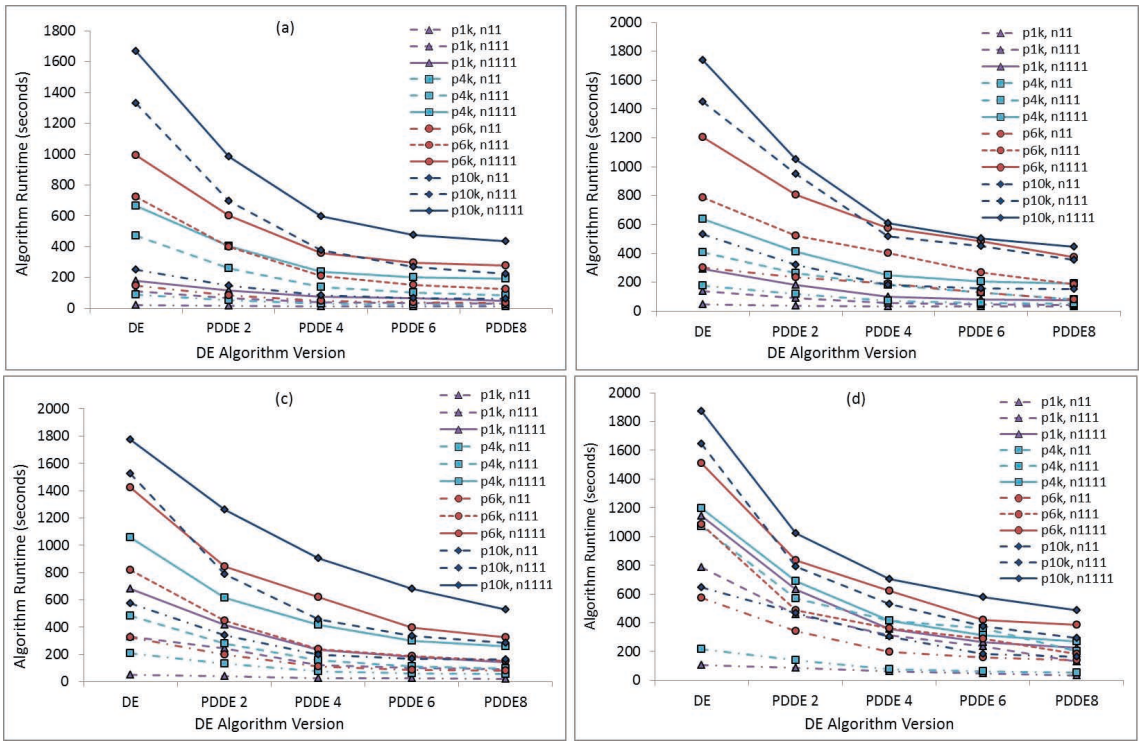

Figure 6. Algorithm Runtimes (seconds) computed by PDDE at 2, 4, 6 and 8 cores with population sizes and number of nodes ranging from 1000, 4000, 6000, 10,000 and 11, 111, 1111 respectively for boundary-value problems, (a) Example 1 (b) Example 2 (c) Example 3 (d) Example 4

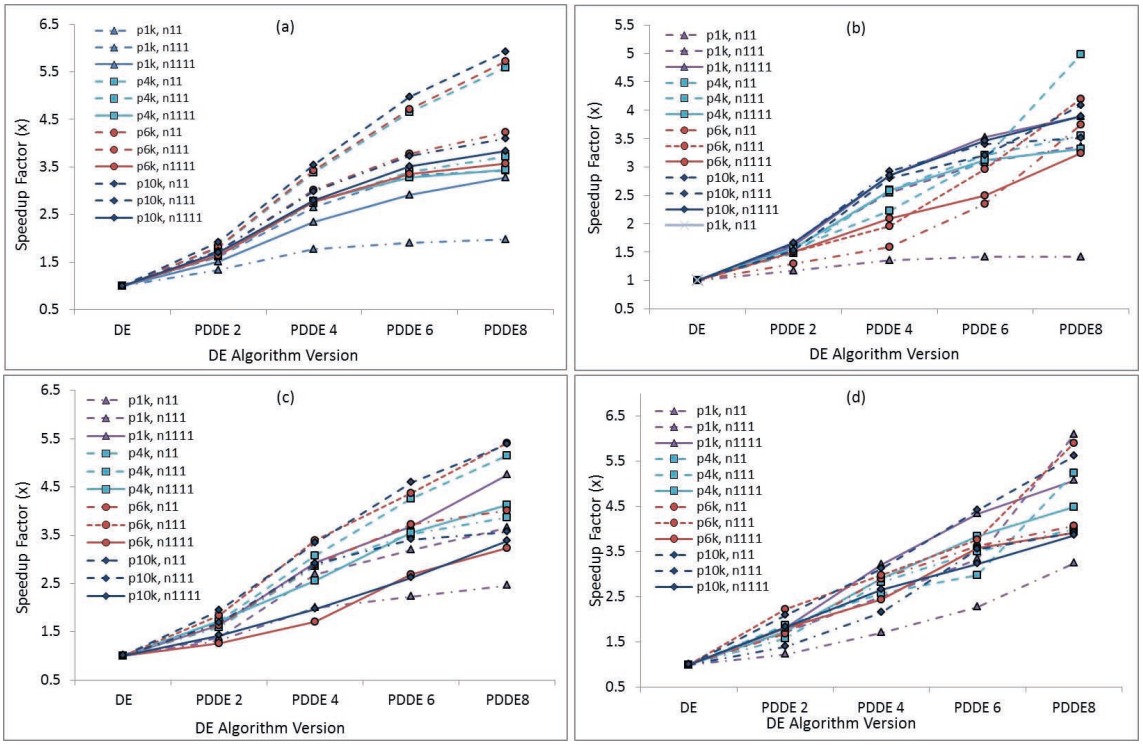

Figure 7. Speedup Factor (x) computed by DE and PDDE at 2, 4, 6 and 8 cores with population sizes and number of nodes ranging from 1000, 4000, 6000, 10,000 and 11, 111, 1111 respectively for boundary-value problems, (a) Example 1 (b) Example 2 (c) Example 3 (d) Example 4

Table 2. Comparison of the exact, DE and PDDE-computed nodal values for Example 1

Accuracy							
Node. No	Exact	DE[20]		PDDE 4 Cores		PDDE 8 Cores	
		yi	di	yi	di	yi	di
1	-3	-3	0	-3	0	-3	0
2	-2.9701	-2.9701888	1.03×10^{-4}	-2.970189	1.03×10^{-4}	-2.97019	1.03×10^{-4}
3	-2.9339	-2.9340842	2.04×10^{-4}	-2.934085	2.04×10^{-4}	-2.934084	2.04×10^{-4}
4	-2.8904	-2.8906887	2.99×10^{-4}	-2.890689	3.00×10^{-4}	-2.890689	2.99×10^{-4}
5	-2.8385	-2.8388643	3.82×10^{-4}	-2.838865	3.82×10^{-4}	-2.838864	3.82×10^{-4}
6	-2.7769	-2.7773139	4.44×10^{-4}	-2.777315	4.45×10^{-4}	-2.777314	4.44×10^{-4}
7	-2.7041	-2.7045598	4.76×10^{-4}	-2.70456	4.77×10^{-4}	-2.70456	4.76×10^{-4}
8	-2.6185	-2.6189208	4.65×10^{-4}	-2.618921	4.66×10^{-4}	-2.618921	4.66×10^{-4}
9	-2.5181	-2.5184852	3.96×10^{-4}	-2.518485	3.96×10^{-4}	-2.518485	3.96×10^{-4}
10	-2.4008	-2.4010807	2.49×10^{-4}	-2.401081	2.49×10^{-4}	-2.401081	2.49×10^{-4}
11	-2.2642	-2.264241	0	-2.264241	0	-2.264241	0

Table 3. Comparison of the exact, DE and PDDE-computed nodal values for Example 2

Accuracy							
Node. No	Exact	DE[20]		PDDE 4 Cores		PDDE 8 Cores	
		yi	di	yi	di	yi	di
1	1	1	0	1	0	1	0
2	0.8187	0.81897551	2.45×10^{-4}	0.8189757	2.45×10^{-4}	0.818975	2.45×10^{-4}
3	0.6703	0.67071004	3.90×10^{-4}	0.6707103	3.90×10^{-4}	0.67071	3.90×10^{-4}
4	0.5488	0.549272972	4.61×10^{-4}	0.5492731	4.62×10^{-4}	0.549273	4.62×10^{-4}
5	0.4493	0.449806823	4.78×10^{-4}	0.4498069	4.78×10^{-4}	0.449807	4.78×10^{-4}
6	0.3679	0.368332946	4.54×10^{-4}	0.3683331	4.54×10^{-4}	0.368333	4.54×10^{-4}
7	0.3012	0.301592386	3.98×10^{-4}	0.3015927	3.98×10^{-4}	0.301593	3.99×10^{-4}
8	0.2466	0.246915522	3.19×10^{-4}	0.2469159	3.19×10^{-4}	0.246916	3.19×10^{-4}
9	0.2019	0.202115279	2.19×10^{-4}	0.2021155	2.19×10^{-4}	0.202115	2.19×10^{-4}
10	0.1653	0.165399647	1.01×10^{-4}	0.1653999	1.01×10^{-4}	0.1654	1.01×10^{-4}
11	0.1353	0.1353	0	0.1353	0	0.1353	0

lation size from 1000 to 10,000 individuals. Figures 6-7 shows that with an increase in data size, the speedup factor achieved at 4 and 8 cores also increases. Hence, the proposed algorithm is scalable with number of cores. Also, we studied the performance of the algorithm with increasing number of nodes from 11 to 1111. The best speedups achieved are for 111 nodes throughout all numerical examples and all population numbers as shown in Figures 6-7.

In the proposed PDDE algorithm for efficient optimization of linear and non-linear boundary value problems, all eight processors on the machine are used to calculate number of threads equal to the number of population individuals to evolve the population variable with increasing number of generations. In case where population size and number of nodes is increased, the problem size becomes large hence a significant speedup is achieved through parallelization.

Generally, the speedup increases when the number of processors increases. In some cases, because of the communication overhead, the increasing speedup is not exactly linear. As a master-slave model parallelizes its individual evaluation tasks as well as some other operations (such as local search) on the slave nodes, the model has an operation-level of parallelism.

Distributed evolutionary algorithms (dEAs) improve the efficiency of EAs, and on the other hand they enhance the global search ability. In this sense, the proposed PDDE technique improves the availability for solving real-world problems with large-scale, high-dimensional, and complex features [24].

4 Conclusion

In this work, we proposed a population-distributed approach to increase the performance of differential evolution algorithm in solving boundary value problems. The performance of distributed differential evolution algorithm is evaluated in terms of a comparison in computation time for sequential and parallel programs; convergence accuracy, speedup and scalability. The data parallelization methodology adopted assigns each population individually, to a separate core. One thread of the computation processed by one core consists of one individual competing with an offspring, evaluating and comparing fitness values for parent and offspring and selection of the fittest individual for the next generation. The computation time and accuracy of the algorithm was evaluated on a multicore platform with 2 to 8 cores for population sizes 1000 to 10,000 individuals and number of nodes 11, 111 and 1111. Comparative testing shows that the average speedup achieved for the four benchmark problems is $3.37\times$ while keeping the accuracy of the algorithm unaffected for sequential and parallel architectures.

Acknowledgement

This work was partially funded by Fondazione Cariverona, Italy.

References

[1] Gong, Y.J., Chen, W.N., Zhan, Z.H., Zhang, J., Li, Y., Zhang, Q. and Li, J.J., 2015, Distributed evolutionary algorithms and their models: A survey of the state-of-the-art, Applied Soft Computing, 34, pp. 286-300. DOI: 10.1016/j.asoc.2015.04.061

[2] Zelinka, I., 2015, A survey on evolutionary algorithms dynamics and its complexity–Mutual relations, past, present and future, Swarm and Evolutionary Computation, 25, pp. 2-14. DOI: 10.1016/j.swevo.2015.06.002

[3] Price, K., Storn, R.M. and Lampinen, J.A., 2006, Differential evolution: a practical approach to global optimization, Springer Science Business Media, ISBN: 978-3-540-20950-8

[4] Storn, R. and Price, K., 1997, Differential Evolution–a simple and efficient heuristic for global optimization over continuous spaces, Journal of global optimization, 11(4), pp. 341-359. DOI: 10.1023/A:1008202821328

[5] Charles, A.J. and Parks, G.T., 2017, Mixed Oxide LWR Assembly Design Optimization Using Differential Evolution Algorithms, 2017 25th International Conference on Nuclear Engineering, Shanghai, China, 9, pp. V009T15A065. DOI: 10.1115/ICONE25-67936

[6] Zaharie, D. and Petcu, D., 2005, Parallel implementation of multi-population differential evolution, Proc. of the NATO Advanced Research Workshop on Concurrent information processing and computing, Nicolau, A. and Grigoras, D., eds., Sinaia, Romania, pp. 223-232.

Table 4. Comparison of the exact, DE and PDDE-computed nodal values for Example 3

Accuracy							
Node. No	Exact	DE[20]		PDDE 4 Cores		PDDE 8 Cores	
		yi	di	yi	di	yi	di
1	17	17	0	17	0	17	0
2	15.7555	15.7554545	5.79 x 10^{-4}	15.754875	5.80 x 10^{-4}	15.754875	5.79 x 10^{-4}
3	14.7733	14.7733333	9.11 x 10^{-4}	14.772422	9.11 x 10^{-4}	14.772423	9.11 x 10^{-4}
4	13.9977	13.9976923	1.07 x 10^{-4}	13.996624	1.07 x 10^{-4}	13.996625	1.07 x 10^{-4}
5	13.3886	13.3885714	1.10 x 10^{-4}	13.387473	1.10 x 10^{-4}	13.387474	1.10 x 10^{-4}
6	12.9167	12.9166667	1.03 x 10^{-4}	12.915631	1.04 x 10^{-4}	12.915632	1.03 x 10^{-4}
7	12.56	12.56	9.04 x 10^{-4}	12.559095	9.05 x 10^{-4}	12.559096	9.04 x 10^{-4}
8	12.3018	12.3017647	7.23 x 10^{-4}	12.301041	7.24 x 10^{-4}	12.301042	7.23 x 10^{-4}
9	12.1289	12.1288889	5.05 x 10^{-4}	12.128383	5.06 x 10^{-4}	12.128384	5.05 x 10^{-4}
10	12.0311	12.0310526	2.61 x 10^{-4}	12.030791	2.61 x 10^{-4}	12.030791	2.61 x 10^{-4}
11	12	12	0	12	0	12	0

Table 5. Comparison of the exact, DE and PDDE-computed nodal values for Example 4

Accuracy							
Node. No	Exact	DE[20]		PDDE 4 Cores		PDDE 8 Cores	
		yi	di	yi	di	yi	di
1	-1	-1	0	-1	0	-1	0
2	-0.8144	-0.8142	1.24 x 10^{-4}	15.754875	1.24 x 10^{-4}	15.754875	1.24 x 10^{-4}
3	-0.655	-0.6548	2.11 x 10^{-4}	14.772422	2.11 x 10^{-4}	14.772423	2.11 x 10^{-4}
4	-0.5186	-0.5183	2.64 x 10^{-4}	13.996624	2.65 x 10^{-4}	13.996625	2.64 x 10^{-4}
5	-0.4022	-0.4019	2.88 x 10^{-4}	13.387473	2.89 x 10^{-4}	13.387474	2.87 x 10^{-4}
6	-0.3033	-0.303	2.85 x 10^{-4}	12.915631	2.86 x 10^{-4}	12.915632	2.84 x 10^{-4}
7	-0.2195	-0.2193	2.59 x 10^{-4}	12.559095	2.61 x 10^{-4}	12.559096	2.59 x 10^{-4}
8	-0.149	-0.1488	2.15 x 10^{-4}	12.301041	2.16 x 10^{-4}	12.301042	2.14 x 10^{-4}
9	-0.0899	-0.0897	1.54 x 10^{-4}	12.128383	1.56 x 10^{-4}	12.128384	1.54 x 10^{-4}
10	-0.0407	-0.0406	8.22 x 10^{-4}	12.030791	8.26 x 10^{-4}	12.030791	8.19 x 10^{-4}
11	0	0	0	0	0	0	0

[7] Ge, Y.F., Yu, W.J. and Zhang, J., 2016, Diversity-Based Multi-Population Differential Evolution for Large-Scale Optimization, Proc. of the 2016 on Genetic and Evolutionary Computation Conference Companion, Denver, Colorado, USA, pp. 31-32. DOI: 10.1145/2908961.2908995

[8] Cheng, J., Zhang, G., Caraffini, F. and Neri, F., 2015, Multicriteria adaptive differential evolution for global numerical optimization, Integrated Computer-Aided Engineering, 22(2), pp. 103-107. DOI: 10.3233/ICA-150481

[9] Lobato, F.S., Steffen Jr, V. and Silva Neto, A.J., 2010, A comparative study of the application of differential evolution and simulated annealing in radiative transfer problems, Journal of the Brazilian Society of Mechanical Sciences and Engineering, 32(SPE), pp. 518-526. DOI: 10.1590/S1678-58782010000500012

[10] Hartfield, R.J., Jenkins, R.M. and Burkhalter, J.E., 2007, Ramjet powered missile design using a genetic algorithm, Journal of Computing and Information Science in Engineering, 7(2), pp. 167-173. DOI: 0.1115/1.2738722

[11] Penas, D.R., Banga, J.R., González, P. and Doallo, R., 2015, Enhanced parallel differential evolution algorithm for problems in computational systems biology, Applied Soft Computing, 33, pp. 86-99. DOI: 10.1016/j.asoc.2015.04.025

[12] González-Álvarez, D.L., Vega-Rodríguez, M.A. and Rubio-Largo, Á., 2014, Parallelizing and optimizing a hybrid differential evolution with Pareto tournaments for discovering motifs in DNA sequences, The Journal of Supercomputing, 70(2), pp. 880-905. DOI: 10.1007/s11227-014-1266-y

[13] Kozlov, K. and Samsonov, A., 2011, DEEP—differential evolution entirely parallel method for gene regulatory networks, The Journal of Supercomputing, 57(2), pp. 172-178. DOI: 10.1007/s11227-010-0390-6

[14] Maciejewski, Ł., 2007, Application of differential evolution algorithm for identification of experimantal data, Archive of Mechanical Engineering, 54(4), pp. 327-337.

[15] Nayak, N., Routray, S.K. and Rout, P.K., 2016, Design of Takagi-Sugeno fuzzy controller for VSC-HVDC parallel AC transmission system using differential evolution algorithm, International Journal of Computer Aided Engineering and Technology, 8(3), pp. 277-294. DOI: 10.1504/IJCAET.2016.077605

[16] Mokhtari, H. and Salmasnia, A., 2015, A Monte Carlo simulation based chaotic differential evolution algorithm for scheduling a stochastic parallel processor system, Expert Systems with Applications, 42(20), pp. 7132-7147. DOI: 10.1016/j.eswa.2015.05.015

[17] Acebrón, J.A. and Spigler, R., 2007, Supercomputing applications to the numerical modeling of industrial and applied mathematics problems, The Journal of Supercomputing, 40(1), pp. 67-80. DOI: 10.1007/s11227-006-0014-3

[18] Tardivo, M.L., Caymes-Scutari, P., Mendez-Garabetti, M. and Bianchini, G., 2013, Two models for parallel differential evolution, Proc. of HPCLatAm, C. Garcia Garino and M. Printista, eds., Mendoza, Argentina, pp. 25-36.

[19] Ntipteni, M.S., Valakos, I.M. and Nikolos, I.K., 2006, An asynchronous parallel differential evolution algorithm, Proc. of the ERCOFTAC conference on design optimisation: methods and application.

[20] Fateh, M.F., Zameer, A., Mirza, N.M., Mirza, S.M. and Raja, M.A.Z., 2017, Biologically inspired computing framework for solving two-point boundary value problems using differential evolution, Neural Computing and Applications, 28(8), pp. 2165-2179. DOI: 10.1007/s00521-016-2185-z

[21] Tasoulis, D.K., Pavlidis, N.G., Plagianakos, V.P. and Vrahatis, M.N., 2004, Parallel differential evolution, Proc. of the 2004 Congress on Evolutionary Computation, Portland, Oregon, USA, pp. 2023-2029. DOI: 10.1109/CEC.2004.1331145

[22] Abo-Hammour, Z.S., Yusuf, M., Mirza, N.M., Mirza, S.M., Arif, M. and Khurshid, J., 2004, Numerical solution of second-order, two-point boundary value problems using continuous genetic algorithms, International Journal for Numerical Methods in Engineering, 61(8), pp. 1219-1242. DOI: 10.1002/nme.1108

[23] Tat, C.K., Majid, Z.A., Suleiman, M. and Senu, N., 2012, Solving Linear Two-Point Boundary Value, Applied Mathematical Sciences, 6(99), pp. 4921-4929.

[24] Zurita, N.F.S., Colby, M.K., Tumer, I.Y., Hoyle, C. and Tumer, K., 2018, Design of Complex Engineered Systems Using Multi-Agent Coordination, Journal of Computing and Information Science in Engineering, 18(1), pp. 011003. DOI: 10.1115/1.4038158

Amnah Nasim is currently doing her Ph.D. in Information Engineering at Universita Politecnica delle Marche, Ancona, Italy. She received her Master's Degree in Computational Sciences and Engineering and has worked as a research assistant from 2015 to 2017 with the Human Factors Research Group at the National University of Sciences and Technology, Islamabad, Pakistan. Since 2018 she is a student Member of IEEE and National Group of Bioengineering. Her main research interest is the automatic processing of digital cardiovascular signals (electrocardiograms), particularly signal processing of cardiovascular data acquired through wearable sensors.

Laura Burattini is a Professor at Università Politecnica delle Marche, Ancona, Italy. She earned her master's degree in electrical/Biomedical Engineering from Politecnico di Milano, Itay in 1993 and Ph.D. in Electrical/Biomedical Engineering in 1998 from University of Rochester, USA. She is the author of more than 60 journal papers and 100 proceedings. She is also the co-founder of B.M.E.D. Bio-Medical Engineering Development srl, an academic spin-off for which she has served as a CEO and President from 2012 to 2016. Her research interests relate to processing and modelling of biomedical data, especially of the cardiovascular, metabolic and motor systems.

Muhammad Faisal Fateh is currently working as a research officer at the National Academy of Young Scientists (NAYS), Islamabad, Pakistan. He received his Master's Degree in Systems Engineering from Pakistan Institute of Engineering and Applied Sciences, Pakistan in 2015. His main research interests are metaheuristic and evolutionary algorithms, boundary value problems specifically in the field of fluid dynamics.

Aneela Zameer is a Professor at the Department of Computer and Information Sciences, Pakistan Institute of Engineering and Applied Sciences, Islamabad, Pakistan. She has a basic degree in Physics from Quaid-e-Azam University, Islamabad, Pakistan in 1997 and Ph.D. from University of West of Scotland, UK on Computational Physics in 2005. She has a Post-Doctoral experience from University of Glasgow in 2007-8. She has a strong research experience in areas of machine learning, artificial intelligence, parallel computing, computational physics and mathematics.