Received: July 16, 2023. Revised: August 6, 2023.

International Journal of
Intelligent Engineering & Systems

http://www.inass.org/

Performance Analysis of Multi-Threaded Distributed Evolutionary Algorithms for Image-Processing Applications

Raghul S[1] **Jeyakumar G[1]***

[1]*Department of Computer Science and Engineering, Amrita School of Computing, Coimbatore Amrita Vishwa Vidyapeetham, India.*
*Corresponding author's Email: g_jeyakumar@cb.amrita.edu

Abstract: Evolutionary algorithms (EAs) in the repository of the evolutionary computing (EC) paradigm are simplistic and stochastic in nature. The desirable characteristic of adaptive simplicity has sustained research community's rampant curiosity in EAs. Distributed evolutionary algorithmic (DEA) framework is an instinctive extension of EAs. This paper summarizes the design and implementation of a multi-threaded DEA (MTDEA) framework. Empirical analysis between a traditional DEA framework and a simulated MTDEA is presented. The MTDEA framework was outfitted with three plug-in algorithms to handle commonly occurring faults in distributed environments. The propriety of the MTDEA framework was validated in two customary applications - Image thresholding and Image reproduction. The thresholding and reproduction problems were addressed by incorporation of the differential evolution (DE) and genetic algorithm (GA) methods respectively, into the MTDEA framework. The performances of DE and GA routines were compared with their sequential counterpart frameworks. The MTDEA framework was evaluated for reliability and scalability on a high-performance computing (HPC) system.

Keywords: Differential evolution, Distributed frameworks, Evolutionary algorithms, Fault tolerant, Genetic algorithm, Image reproduction, Image segmentation.

1. Introduction

Motivated by their metaheuristic characteristics, evolutionary algorithms (EAs) have drawn the attention of researchers around the world, to solve optimization problems across diverse domains. EAs utilize naturally evoked mechanisms to tackle optimization problems through processes that mimic living organisms' natural behaviors. Given a specific optimization problem, EAs generate a random solution space at the outset, followed by iterative generation of new solutions from existing solutions, as well as searches for optimal solutions. Nevertheless, EAs search-based problem-solving methodology has certain limitations such as convergence towards local minima and restricting the search space towards the local optima, which renders them ineffective in taking on complex multi-objective optimization problems with higher dimensions [1].

Pursuant to a decade of active research attempts to overcome EA's drawbacks, the evolutionary computing (EC) research community has been looking into parallel and distributed computational approaches. Unification of EAs into parallel and distributed computing (DC) systems has been propitious in preference to sequential methods, towards solving complex optimization problems.

Recent times have witnessed unabated exponential advancements in the application of image processing techniques in the domains of science and technology. The broad range of appliances include remote sensing, medical imaging, acoustic imaging, and biometric identification. Development of algorithmic frameworks producing optimal solutions with minimal computation complexity, for image processing, is in high demand in the distribution communication-based industry. Population-tweaked EAs, marshalled by their parallel and distributed extensions, have been remarkably sought after investigative avenues for image processing problems. This paper zeroed in on two everyday image processing applications - image segmentation and image reproduction - using distributed evolutionary computing (DEC) frameworks.

Image thresholding is a kind of image segmentation that segregates an ambient scenario into background and foreground images. Image thresholding tasks play crucial roles in image processing routines, there exist various image thresholding techniques [2] in the literature. The typical Otsu method [3] processes a simple image to provide the required segmented image, except when Otsu is exposed to an image with minimal signal-to-noise ratio. Otsu being highly sensitive to noisy images, image segmentation is preceded by image-smoothening. Albeit the '*2D (two-dimension) maximum between cluster variance*' [4] scheme does the job, the time complexity of implementing the algorithm increases proportionately. Published literature is rife with diverse EA-based image thresholding approaches. Differential Evolution (DE) is one of the EAs well known for its simple stochastic nature. Proficient DE has a proven track-record dealing with demanding image processing encounters. The evolutionary nature of the DE algorithm can be exploited for threshold selection for a given set of noisy input images. Decomposition of input image into sub-images that are segregated via DE-based thresholding modulates peculiar over-segmentation to provide the pertinent resultant image.

Commonplace image reproduction and enhancement processes are intended to create a copy of a given image, with a picture quality same as the original, or superior to it. Genetic Algorithm (GA), a member of the EA family, has been actively adopted by researchers for these processes [5].

This paper zeroes in on the design of pertinent DEC frameworks to solve the dual problems of image segmentation and image reproduction, A multithreaded differential evolution algorithm (MTDEA) framework was implemented as a base platform, involving the algorithmic strategy presented in [6]. Incorporation of three plug-in algorithms [7] capacitated MTDEA's resilience to three commonly occurring faults in DC scenarios. The proffered MTDEA framework was validated empirically, via a benchmarked study. MTDEA's effectiveness in tackling image segmentation and image reproduction problems were then studied, incorporating DE and GA algorithms, respectively. Further, to test the framework's complexity, the image of size 1.5 GB from European Southern Observatory (ESO) website was taken as sample and tested. The average execution time is taken as a crucial performance metrics to provide the comparison between the sequential model and the MTDEA model. The MTDEA model outperformed the traditional sequential model serving the targeted goal of decreased execution time devoid of sacrificing to the solution quality.

The remaining parts of this paper are organized as follows – Section 2 presents an algorithmic summary of EAs employed in this investigation, Section 3 covers a review of related works, Section 4 details the MTDEA framework, Section 5 presents the design of experiments for the appraisal of the MTDEA frameworks, followed by discussion of the results, Section 6 reports on solving image processing problems using the MTDEA frameworks. Section 7 presents the performance of the MTDEA framework in an HPC system. Finally, Section 8 includes closing remarks as a conclusion, highlighting future directions.

1.2 Notations used in the work

Variables	Description
NP	Population size
D	Dimension
Cr	Crossover rate
F	Mutation factor
$X_{i,G}$	Candidate vector
$V_{i,G}$	Mutant vector
$U_{i,G}$	Trial vector
t	Threshold value
L	Level of grey color present
w_i	Number of pixels present
u_i	Average value of pixels present
$w_n(i)$	Number of pixels in the category
$u_n(i)$	Average value of pixels in the category
f	Threshold fluctuation factor

2. Evolutionary algorithms in the experiments

This work primarily evaluated the propriety of unifying different EAs with the base MTDEA framework, to assess their applicability to various image processing problems. DE and GA were the algorithms considered for the experiments to solve the image segmentation and image reproduction problems, respectively. Skeletal views of these algorithms are presented in this section.

2.1 Differential evolution (DE) algorithm

The DE algorithm, which was introduced to the EC research community, has proven to be a highly stochastic mechanism for real parametric optimization [8]. DE's effectiveness has been ratified by its application to numerous benchmarking and optimization problems in the real world. Fundamentally, DE is a population-centric optimization algorithm that utilizes different variations and selection mechanisms, similar to other EAs like estimation of distribution algorithms, evolution strategies and genetic algorithms. However, DE's unique characteristic of innate Differential Mutation distinguishes it from other EAs. DE's differential mutation operator aggregates the scaled difference of two solution vectors to another vector, to generate a mutant vector as an interim solution vector. DE then employs a recombination mechanism between the interim and its mutant vectors, to evoke a target vector. Next, a selection mechanism is employed to designate the survivor for the next generation, by comparing the interim vector and the trial vector.

DE is a search-based algorithm that explores the search space of the targeted optimization problem using various sampling methods, by the randomized selection from a population size of $\boldsymbol{NP}$ vectors with D dimensions. $\boldsymbol{NP}$ as an initial population, and each individual within it is a candidate solution ($X_{i,G}$) as denoted in Eq. (1),

$$\boldsymbol{X_{i,G}} = \{\mathrm{x}_{\mathrm{i,G}}^{1}, \ldots, \mathrm{x}_{\mathrm{i,G}}^{\mathrm{D}}\}\ \mathrm{i} = \mathrm{i}, \ldots, \boldsymbol{NP}. \tag{1}$$

The original (initial) population is set to cover the given search space efficiently, by random generation of vectors throughout the search space. An iterative process is launched once the initial population is ready to generate new populations, until the termination condition is satisfied. At every iteration (generation), a new mutant vector ($V_{i,G}$) is generated, as shown in Eq. (2),

$$\boldsymbol{V_{i,G}} = \{\mathrm{v}_{\mathrm{i,G}}^{1}, \ldots, \mathrm{v}_{\mathrm{i,G}}^{\mathrm{D}}\}\ \mathrm{i} = \mathrm{i}, \ldots, \boldsymbol{NP}. \tag{2}$$

for each individual (the target vector, $\mathrm{X}_{\mathrm{i,G}}$) present in the current population. Diverse mutation strategies proposed in the public domain, such as DE/rand/1, DE/best/1 and DE/current-to-best. [9, 10, 11], differ in the manner by which the mutant vectors are generated. Subsequent to mutation, the crossover (which may be binomial or exponential) is performed between the mutant vector, $\boldsymbol{V_{i,G}}$, and the target vector, $\boldsymbol{X_{i,G}}$, to produce the trial vector ($\boldsymbol{U_{i,G}}$) as shown in Eq. (3),

$$\boldsymbol{U_{i,G}} = \{u_{i,G}^{1}, \ldots, u_{i,G}^{D}\} \tag{3}$$

DE's desirable traits of ease of implementation, algorithmic simplicity, and fast convergence have captivated the research community. Over the past two decades, active research in fortification of the DE algorithm have yielded numerous DE variants. Selection of an appropriate DE variant for a specific optimization problem continues to pose an arduous exercise for the research community, few of the investigation attempts in this direction have been summarised in [12].

2.2 Genetic algorithm

J. H. Holland brought to light the significant concept of utilizing GA to solve optimization problems, in 1970 [13]. Since then, GA has galvanized the global research community to take on optimization problems in scientific and engineering domains. GA has not been touted as a mathematically sound algorithm, as the retrieved optimum solution is evolved in each generation, without rigorous mathematical formulation, unlike conventional gradient-type optimization. GA functions effectively with all types of solution representations. A population pool comprised of solution vectors is initialized at random, and then updated, iteratively. In every iteration of the genetic process, a new solution vector is generated from the vectors that were present in the previous generation. This evolutionary process can be effective only if a specific selection routine is chosen to select the parents with the best fitness. GA embodies a variety of mutation and recombination operators to produce offspring. This process of evolution or gene manipulation is anticipated to produce better children that survive in subsequent generations, imitative of the concept of survival-of-the-fittest. The procedure to find the best candidate is repeated until the termination condition is satisfied.

3. Related works

This section reviews works reported in the published domain apropos of the parallel and DC nature of EAs, which afford meaningful insights into the image segmentation and image reproduction algorithms.

Bharathi et al., [14] provided a comprehensive survey on EAs and their applications. Incorporation of EAs in parallel and DC frameworks have turned in effective performance in the manipulation of run-of-the-mill sequential use cases. Load distribution and high computation speed can be accomplished by parallel processing and distribution of` the tasks among various computing elements. The island, hierarchical, Master-Slave, Cellular, pool, multi-agent, and Co-evolution are few examples of the widely employed parallel and distributed evolutionary models [15]. The authors in [16] proposed an agent-based approach, which can solve an optimization problem by decomposing it into constituent sub-problems. These agents (master and slaves) are organized into groups at the first level of the agent-based approach, prior to initiation of the optimization process. Said et al., [17] proposed a hybridized approach, combining GA with an estimation of distributions. In this hybrid framework, the master node exploits the search space, whereas the slave nodes perform specific proportionate functions required for the intended optimization.

A collective communication-based approach was proposed by Abdoun et al., [18]; communications between master and slave nodes take place via message passing in java (MPJ) express. Abdoun's team utilized an Ant Colony Optimization Algorithm to address the optimization problem. Similarly, Depolli et al., proposed an asynchronous master-slave, user-defined, framework to solve time-intensive multi-objective optimization problems [19], where the settings and input parameters are formulated based on the targeted optimization problem. Utilizing partitioned global space address (PGAS) for connecting computers as a cluster in a virtual distributed form, a high level of parallelism was achieved in [19]. Lin et al., [20] put forth a parallel GA (PGA), where every node computes the initial population and shares the best candidate with all the extant nodes, creating multiple work areas. The authors in [21] used the PGA concept of [20] to take on non-linear optimization problems.

In [22], the author proposed utilization of shared storage space and centralized load-balancing principles, to solve multi-objective optimization problems. Zhan et al., [23] submitted a heterogeneous EA approach with a double-layer architecture; an EA is equipped in the first layer; the cloud paradigm is introduced in the second layer, to provide distributed environment. Similarly, Supriya et al., proposed a Bayesian learning approach [24] and demonstrated dynamic resource provisioning, for cloud-based applications. Cao et al., introduced a distributed parallel cooperative co-evolutionary multi-objective evolutionary algorithm (DPCCMOEA), to solve large-scale optimization problems [25]. DPCCMOEA decomposes the original large-scale problems into sub-problems and introduces a two-layer message-passing architecture that evolve around the sub-problems. This approach works on the principles of the decision variable analysis (DVA) strategy. Karthi's team conducted a detailed performance analysis of GA for functional optimization in a multi-core platform [26].

Albeit multiple pieces of published literature advocate the advantage of parallel and distributed systems, competition-based [27] and Divide-conquer-based [28] methods have also aroused keen interest among researchers. These popular schemes are generally used to tackle optimization problems.

EAs are widely combined to address diverse image processing applications. Michal et al., utilizing GA, submitted a fast-robotic pencil drawing application [29]. Sun, Yu, et al., [30] propounded an image registration technique with the DE algorithm. Liu et al, recommended a collaborative dragonfly algorithm with a novel communication strategy, for the segmentation of multi-thresholding color images [31]. Meesala et al., proffered a feature-based emotion detection in social media, using multi-objective EAs [32]. Luo's group recommended a Whale Optimization Algorithm to reduce noise present in an image [33].

Image segmentation is crucial in medical image processing and related domains. Cuckoo search, incorporated with Masi entropy and DE mutation, was presented to handle multi-level image segmentation scenarios [34]. Akshay et al., [35] suggested an image segmentation technique via an artificial bee colony algorithm, manipulating gray level co-occurrence matrix (GLCM) features, for categorization of fruit images. Sun's team [36] recommended an adaptive bi-mutation-based DE algorithm for image segmentation, outfitted with the opposite learning strategy, to improve the quality of the initial population. In the same probe, Sun's team proposed a threshold-value-Jia based mutation strategy, to enhance the explorative capability of the algorithm. Bhandari presented a novel fast multi-level thresholding beta-DE algorithm, for segmentation of color images [37]. The author calculated the optimum threshold by maximizing the Kapur and Tsallis entropy [38], a thresholding function incorporated within the beta-DE algorithm. Tarkhaneh and his team [39] proposed an adaptive DE algorithm for optimal multi-level thresholding, for the segmentation of MRI images of brain scans. The authors compared the proposed algorithm against the native DE algorithm on three DE benchmark algorithms, with T2-weighted MRI brain images. Jia's group [40] proposed a new hybrid algorithm, blending the grasshopper optimization algorithm with DE, for segmentation of multi-level satellite images.

Similar to image segmentation, literature is rife with numerous studies that manage problems apropos of image reproduction and image enhancement.

Table 1. Algorithmic description of the MTDEA framework

```
In Master node

    Create Initial population - pop( ).
    Based upon the number of nodes decompose
        the pop( ).
    Send the decomposed population to the
        corresponding nodes.
    After every iteration receives the best candidate
        from all nodes.
    Compare and update the best candidate until
        termination condition is satisfied.

In slave node

    Receive the population from master node -
        Pop_1( ).
    Decompose the Pop_1( ) based on the number
        of threads.
    Send the decomposed population to the
        corresponding threads .
    After every iteration - compare the best
        candidates between the threads.
    If (iteration count / mf  == 0)
    // mf - migration frequency
        send the best candidate to (n+1)th node.

In Threads

    Receive sub-population from the corresponding
        slave node.
    Implement the specified EA for the population.
    After every iteration send the best candidate to
        the slave node.
```

Kishore’s team proposed a GA-based scheme to regenerate images from shrunk scaled images, using bit data count [41]. Kishore et al., asserted that their algorithm could be applied to recover error bits within any data block. Shrivastava et al., [42], proposed a GA-evoked image-enhancement technique, in which de-noising of the image was accomplished by reduction of the mean noise present in the targeted image. GA-based image segmentation and reconstruction was implemented in [43]; its author crafted a GA scheme to extract a mask that removed and reconstructed the segmented image, from the original image. Similarly, Hashemi et al., [44] proposed a GA application for enhancement of an image, using the intensity value and the number of edges, as fitness measures, for each chromosome.

On appreciation of the imperative of parallel and distributed facets of conventional EAs, and the relevance of algorithmic frameworks for the handling of complex image processing problems, the objective of this paper is to elucidate, implement, and validate the propriety of MTDEA frameworks for image processing applications. Multithreaded distributed differential evolution (MTDDE) and multithreaded distributed genetic algorithm (MTDGA) were the two frameworks assessed in this work.

4. MTDEA framework

Computation time and computational resources assume a vital role in the management of intricate real-world problems. Building an asynchronous multi-threaded MTDEA model over the distributed master-slave model is one of the key contributions of this investigation. This framework expands DEA framework to its multithreaded version, in which each computing element in the distributed framework is endowed with multiple threads (threads are the simplest flow of activities that can be executed in a process) to reinforce the computational power. The suggested framework utilized the “*multiprocessing*” library to provide process-based parallelism; this package supports a spawning process via an application programming interface (API). The goal of the recommended MTDEA framework is to diminish computational time without detriment to the solution quality.

The MTDEA is a highly distributive, fault-tolerant framework that employs an island-based dispensed model. This island model originated from the concepts of natural evolution, which tend to systematize the original population spread throughout the search space. Generally, the number of islands in island models, is user-specified, where the initial population is segregated into sub-populations. Each island contains a sub-population to ensure the principle of simultaneous exploration of various landscapes. Significantly, within each island, the MTDEA framework attempts to sustain high population diversity. In the notion of attaining anticipated behavior, the best candidate from one island is exchanged with the best of other islands, at a specific frequency (*Migration Frequency* (MF)) [45]. The process of exchanging the best candidates is termed Migration. Among the various migration topologies present, best candidates can be chosen by the algorithmic flow of the targeted optimization problem. Table 1. presents an algorithmic description of the MTDEA framework.

The MTDEA framework follows a traditional master-slave architecture for its processing. In the master node, the initial population is generated; based on the number of slave nodes, the initial population is distributed in such a way that all the slave nodes have an equal number of candidates. Every slave node present in a MTDEA framework is an island.

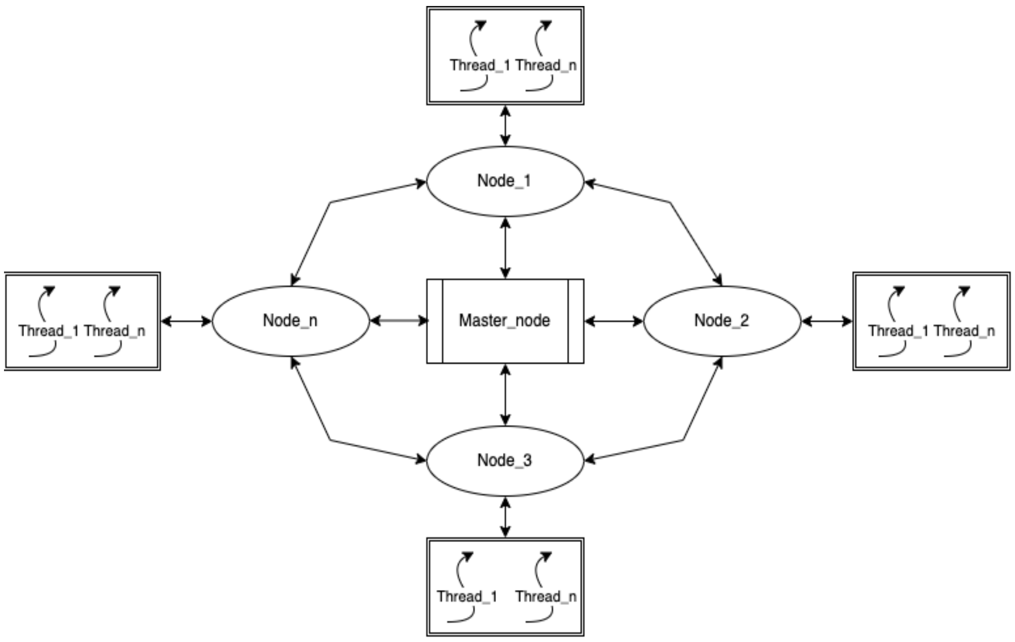

Figure. 1 General architecture of the MTDEA framework

Each slave node receives a sub-population. Furthermore, the decomposition of the sub-population takes place and sent to the respective threads. The number of threads on an island is user-specified, and each thread consist of an EA implied to it. In the MTDEA framework, the threads are administered by the corresponding slaves. Every thread runs the implied EA in a parallel standpoint maintaining the synchronization.

In the implementation part, the threads are designated to run in a multi-threaded mode using the *concurrent.futures.ThreadPoolExecutor( )* API. In this function, *concurrent.futures* module offers a high-level interface for execution of threads in a parallel fashion. The *ThreadPoolExecutor* class has methods to generate pool of threads and to execute them asynchronously. Fig. 1 presents a general architecture of the MTDEA framework.

The MTDEA framework is customizable and scalable based on the natural behavior of a given optimization problem. The acronym MTDDE denotes the MTDEA framework in which the DE algorithm is placed in the slave nodes, with their corresponding threads; MTDGA is the MTDEA framework with GA in the slave nodes with their threads. To provide load balancing between the threads of a corresponding slave node, the initial population of the slave node is decomposed equally based on the number of threads present in the slave node, the initial population of the slave node is decomposed equally based on the number of threads present in the slave node. During every generation, the main objective of each thread is to find the optimal solution for the given sub-population and report the best candidate to the slave. During a specific frequency of generations, the slave nodes are permitted to migrate their best candidates to the neighboring islands, demonstrating a strict migration topology. In the MTDEA framework, ring topology is deployed to maintain a highly cooperative evolutionary environment.

4.1 Fault-tolerant MTDEA framework

In continuation of the design of MTDDE and MTDGA frameworks, experiments were carried out to furnish the MTDDE framework with add fault-tolerant mechanisms. Albeit parallel and or DC systems are employed to work out large-scale optimization problems, the probability of fault-occurrence is higher in these systems. The ability of a framework to deliver its designated functionality despite errors/faults that occur in hardware or software sub-systems is termed fault tolerance [46]. Commonly occurring faults in distributed systems are node failure and link failure. ‘n-version, epidemic, rejuvenation, and addition of checkpoints’ are a sample of the fault tolerant strategies employed for distributed systems [47, 48, 49]. These types of counter-algorithms handle fault instances at the price of increased time complexity of the algorithms. Since user-level failure mitigation proposals (ULFM) are implicit in the message-passing interface (MPI) forum, this work explores the efficiency of simple plug-in algorithms that overcome commonly occurring faults.

Table 2. *MOV* and *Aet* for DDE and MTDDE

	f_1		f_2	
	DDE	MTDDE	DDE	MTDDE
MOV	7.75E-14	3.31E-14	5.82E-14	6.16E-14
Aet (s)	1.634	1.188	1.402	0.68
	f_3		f_4	
	DDE	MTDDE	DDE	MTDDE
MOV	1.3E-05	1.3E-05	1.84E-13	0.00
Aet (s)	86.594	83.8	8.4	2.417

As delineated in [7], three fault-tolerant algorithms are plugged into the MTDDE framework to manage the reported faults, such as – ‘Single-Node failure,’ ‘Multi-Node failure,’ and ‘Communication-Link failure’. The ‘Single-Node failure’ is handled by the addition of a backup node for each slave node. The slave nodes share their population with the backup nodes, after every generation. The MTTDE framework uses a *concurrent.futures.shutdown( )* method, in which the user specifies the wait time. In case of no response from a slave node within this specified wait period, the MTDDE framework shuts down that particular node, assuming there may be a fault in the node. The framework initiates a backup node and starts the communication between the neighboring nodes and the respective slave node.

In a highly distributed data processing environment, where number of computational nodes are high, there is an increased likelihood of a fault event observed as concurrent failure of multiple nodes. After every successive generation, the slave nodes are programmed to regularly send the best candidates to the master node. A master node persistently monitors its slave nodes. If a slave node does not send its best candidate for a certain period of time, which is user specified, the master node assumes the corresponding slave node is in a dead state. Master node initiates the *shutdown( )* method for the dead node. The best candidates received from inactive slave nodes are pooled into a new population within the master node, and the DE algorithm is initiated. Finally, at the end of the process, the master node compares the final optimal solution received from the active nodes with its self-processed optimal solution to dispense the global optimum solution.

In the MTDDE framework, communications between the slave nodes are established by the MPI. There is a strong possibility of communication failure arising due to the natural negotiating process of the MPI. To confront the situation, a trigger is customized in all the slave nodes, such that if MPI provides negative acknowledgment when communicating with the nth node, the trigger is proved. If the trigger is evoked, the sending node starts communication with the (n+1)th node, neglecting the presence of the nth node. This is how the plugged-in algorithm handles the ‘*Communication-Link failure*’ fault.

5. Empirical validation of the MTDEA framework

The initial part of the experiment is framed to compare the performance of the MTDDE framework (the MTDEA framework incorporated with the DE) with the conventional distributed differential evolution (DDE) framework [50]. The experimental setup is noted below:
Laptop PC (MacBook Pro - 2019) with
memory (8 GB 2133 MHz LPDDR3),
graphics card (Intel Iris Plus Graphics 1536 MB), and
processor (Quad-Core Intel Core i5-1.4 GHz).
MTDDE framework threads were customized DE/best/1/bin variant. The control parameter set up with similar values for both DDE and MTDDE frameworks

Population Size (*NP*) = 60,
Crossover rate (*Cr*) = 0.5,
Scaling factor = 0.2,
Dimension (*D*) = 30,
maximum number of generation (*Max_Gen*) = 3000,
number of runs (*nr*) = 30,
migration frequency (*mf*) = 45
migration topology (*mt*) = ‘Ring Topology’.
The termination criteria of this framework are fixed to a minimal value of 1×10^{-12}.

Following four benchmarking functions are added in the experimental setup.

f_1 - Schwefel’s Function 1.2,
f_2 - Rosenbrock’s Function,
f_3 - Generalized Schwefel’s Function, and
f_4 - Ackley’s Function [49, 51]

The performance metrics used to compare the frameworks are average execution time (*Aet*) and mean objective value (*MOV*). The experiments were repeated for different dimensions (*D* = 100, 500, and 1000) to test the MTDDE framework's fault-tolerant capability.

Table 2 presents the *MOV* and *Aet* values for 30 independent comparative runs of the DDE and MTDDE framework.

These results make plain that the Aet values of the MTDDE framework outperform the counterpart DDE variants for all the benchmarked functions, $f_{x\ [x\ =\ 1,2,3,4]}$. MOV metrics reveal that the MTDDE outperforms DDE functions f_1 and f_4. Their performances are similar to the Generalized

Schwefel's Function (f_3). However, the DDE variants overran MTDDE for function f_2.

This simple study demonstrated that multithreaded versions of the DDE frameworks arrive at a faster convergence, ensued by their massive parallel configuration. This comparative study substantiated that the fault-tolerant strategies bestowed on MTDDE facilitate the framework's resilience, - despite the simulated fault conditions, forestalling computational overheads or degradation of quality. The efficiency of the MTDEA framework was validated in terms of serving the goal of reducing the execution time without sacrificing the solution quality.

6. Performance analysis - MTDEA image-processing

The MTDDE and the MTDGA frameworks were capacitated to resolve image segmentation and image reproduction problems, respectively. This section demonstrates the MTDEA framework's prospects in dealing with image segmentation and image reproduction.

6.1 Image segmentation

The process by which a graphical image is partitioned into its constituent subgroups (image segments) is termed as image segmentation. These segments modulate the complexity of a targeted image, to simplify subsequent processing and analysis. The segmentation process allocates labels for every pixel in the image; pixels belonging to same segment possess a unique label.

6.1.1. Background study

Published literature is abuzz with schemes for image segmentation. A traditional straightforward technique is Otsu's method [3]. Conventional Otsu technique delivers an acceptable segmentation if the image is free of noise. Otsu helps in the selection of threshold values, a major challenge posed by image segmentation. The basic idea behind the Otsu method is to divide the image into two categories, A_0 and A_1, based on the threshold value (t). A_0 contains pixel values ranging from [0 to t], and A_1 has pixel values ranging from [(t+1) to (L-1)], where L is the level of grey color present in the targeted image. Otsu is calculated between the categories using the formula given in Eq. (4).

$$\sigma(t)^2 = w_1(t)w_2(t)(u_1(t) - u_2(t))^2 \qquad (4)$$

where, $w_i(t)$ represents the number of pixels present in A_0 and A_1 and $u_i(t)$ represents the average value for all pixels present in A_0 and A_1. The value of t can range from [0 to (L-1)]; the general principle of the Otsu method relies on calculating the variance. Variance is the optimum threshold that can be obtained for a given image. Based on the value of the variance, each pixel is segregated into the foreground and background pixels.

Table 3. Algorithmic description of selecting threshold based on DE

```
Step – 1 : Image pre-processing: Convert RGB image
           into grey scale image
Step – 2 : Population initialization and calculate fitness
           of each individual
Step – 3 : Customized Mutation strategy
           //explained in section 2.1
Step – 4 : Customized Crossover strategy
           //explained in section 2.1
Step – 5 : Selection operation : Comparing the trail
           vector and test vector, candidate with higher
           fitness is selected for forth-coming
           generation
Step – 6 : Check for termination condition
               If termination condition == true:
                       Move to step (7)
               Else:
                       Move to step (3)
Step – 7 : The best candidate's threshold value is
           mapped in-between the range [0 to 255]
Step – 8 :  Search based on [ t-f  to t+f ] will produce
           optimal threshold value
Step – 9 : Based on the optimal threshold value
           obtained, each pixel is segregated into
           foreground and background pixel and the
           final image is produced
```

6.1.2. DE for image segmentation

As part of the pre-processing in image segmentation, the original image is transformed into a grayscale image. The value of each pixel in the grayscale image ranges between 0 and 255. The gene values in the candidates of DE's randomly generated population range from [0.1 to 0.9], to evade the cumulative value of the pixel divided by zero.

The fitness function stated in [52], presented in Eq. (5), is used to evaluate the candidates of DE's population, generated at random.

$$\cos t\,(i) = w_1(i)w_2(i)(u_1(i) - u_2(i))^2 \qquad (5)$$

where, i represents a random vector from the population range [0.1 to 0.9], $w_n(i)$ represents number of pixels in the category (foreground and background) $u_n(i)$ represents the average value of pixels in the category.

As part of the mutation strategy, two vectors are selected at random, from the initial population, and their difference is computed. The mutated vector is obtained by accumulating the candidate with the result.

Apropos of the crossover strategy, a vector is chosen at random from the DE population, such that the selected vector's fitness is not lesser than the target vector; this random vector acts as a cross-cutting object. Implementing a crossover strategy in this fashion affirms convergence rate and population diversity. After n iterations, DE finds the optimal threshold value (t). The search operator focuses on the search space within the specified range of [($t - f$) to ($t + f$)] in successive iterations, where f is the threshold fluctuation factor. Table 3 presents an algorithmic description of selecting a threshold based on DE.

6.1.3. MTDDE for image segmentation

In MTDDE, the master node converts the received original image into a greyscale image. Next, the master node decomposes the greyscale image into equal number of vertical or horizontal slices. The number of slices equals the number of slave nodes present in the model. Each slave node maintains its threads and provides coordination between the threads present in the corresponding slave node. The slave node is responsible for further decomposition of the image into equal halves based on the number of threads present. Each thread is comprised of a DE algorithm incorporated within it; based on the given slice of image, each thread starts to produce the population, at random, and proceeds to discover the optimum threshold value (t) for the given image slice. The settings of the algorithm are as follows:

population size (NP) is 5,
crossover factor (Cr) is 0.3, and
mutation factor (F) is fixed at 0.5.

Maximum number of iterations is set in the range of (10 to 20) generations (to avoid over-segmentation issues, which generally varies from image to image).

With the required parameters, each thread derives the optimal threshold value, which is used to segment the sliced image received from the slave node. The segmented image slice is passed to the slave along with the optimal threshold value (t). During the process of migration (of best candidates), the threshold value of t from each slave node is exchanged with the neighboring slave. Each slave node updates the 't' received from its threads after every independent run. The migration topology used is the ring topology. As the underlying concept of the MTDDE framework is to decompose an original image into smaller regions, based on the region of interest, the optimal threshold is calculated using DE algorithm, which enables image segmentation. When the maximum number of iterations is reached, the slave node collects the segmented images from the threads and merges them into a final segmented image. The master node collects the final segmented images from the slave nodes, and combines them to produce the final segmented image as an output. This approach of decomposing the image into smaller regions and segmenting based on the region of interest, has proven to yield enhanced segmentation quality, besides conserved execution time. One of the observed limitations of the MTDDE framework is over-segmentation that ensues from decomposition of the original image into smaller slices; hence, during the image decomposition process, a reasonable size for slicing should be fixed. Fig. 3 depicts working of the MTDDE framework, containing 1 master node, 1 slave node, and 3 threads.

A sample image was retrieved from the human recognition dataset from the Kaggle repository, for comparative empirical analysis between sequential DE (SDE)-based image segmentation and the MTDDE framework-based image segmentation. To achieve a reasonable analytical study, both these models were customized with the same parametric values. The resultant output images of these models are depicted in Fig. 2. The SDE algorithm and MTDDE framework were deployed for 10 independent runs, and their performance is presented in Table 4. The results are comprised of the optimal threshold achieved by both models, and the corresponding execution times.

NOTE: *SET* – sequential execution time, *DET* – distributed execution time, in Table 4.

6.2 Image reproduction

Image reproduction is an iterative process of reconstructing an original image from its pixel values. It enhances an image to provide a better image for a specific application. There are different techniques to reproduce an image from its original version; one of the popular techniques is by finding similarity measures after every iteration. This process is widely used in reverse imaging, image restoration, and medical images.

Received: July 16, 2023. Revised: August 6, 2023.

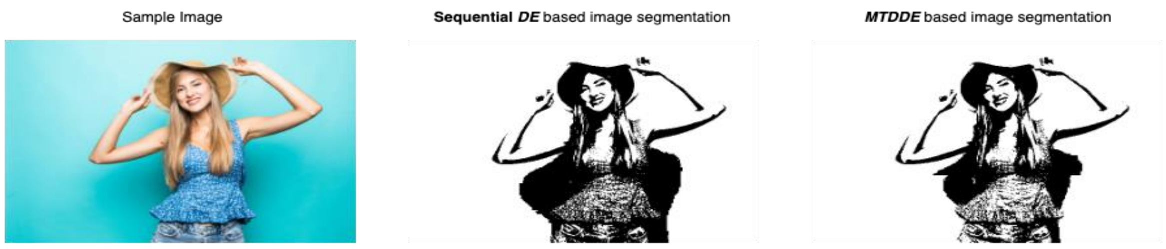
Sample Image
Sequential DE based image segmentation
MTDDE based image segmentation

Figure 2. Resultant output images of image segmentation

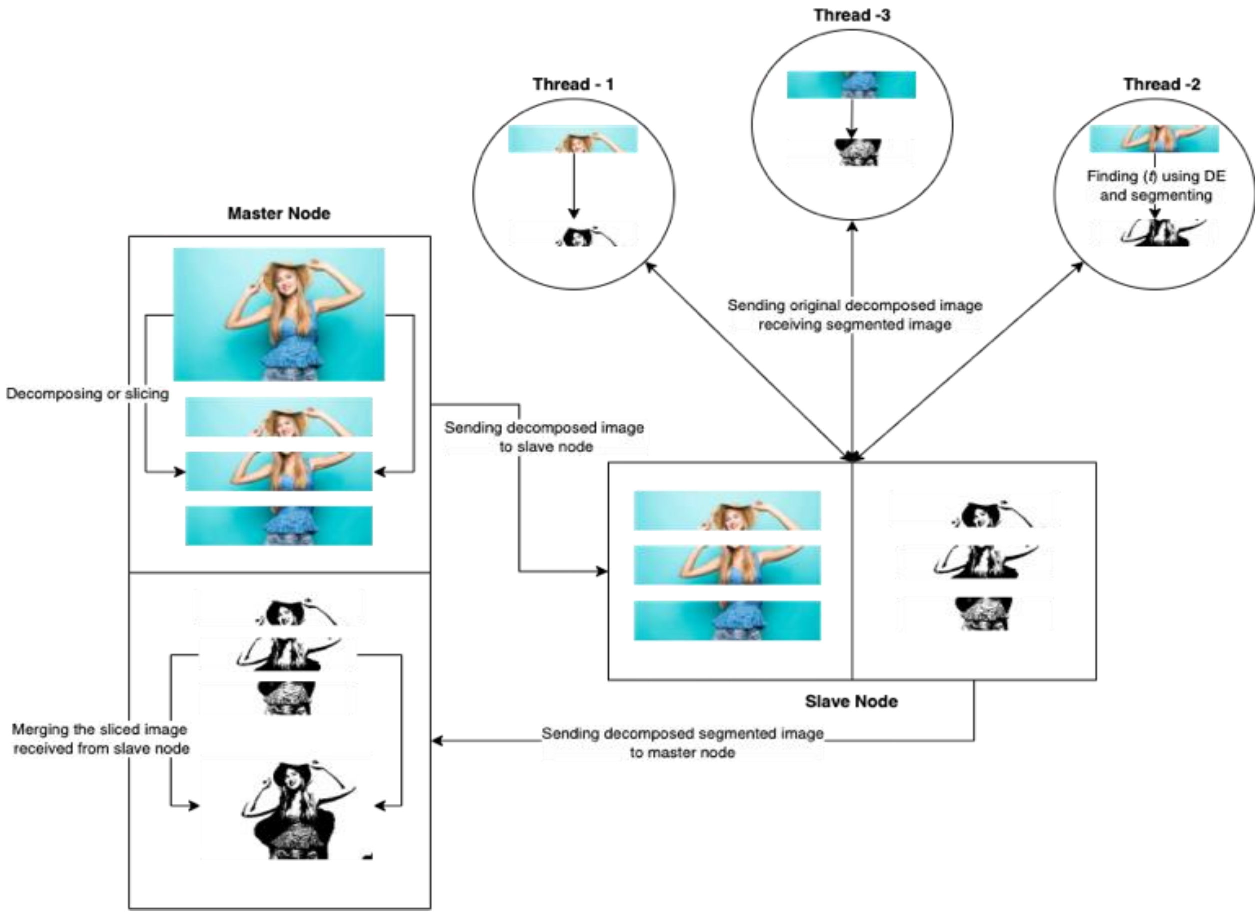
Thread - 1
Thread -3
Thread -2
Finding (t) using DE and segmenting
Master Node
Decomposing or slicing
Sending original decomposed image receiving segmented image
Sending decomposed image to slave node
Slave Node
Merging the sliced image received from slave node
Sending decomposed segmented image to master node

Figure 3. General workflow of MTDDE framework for image segmentation

Table 4. Comparative empirical analysis of SDE and MTDDE

Runs	Optimal Threshold by SDE	SET (s)	Optimal Threshold by MTDDE	DET (s)
1	168.71	0.64	147.56	0.27
2	169.36	0.63	165.84	0.26
3	169.10	0.61	166.87	0.28
4	169.21	0.63	165.47	0.25
5	167.81	0.59	166.37	0.25
6	169.48	0.63	166.92	0.25
7	169.31	0.59	166.91	0.26
8	168.57	0.63	170.03	0.29
9	168.83	0.63	167.02	0.29
10	168.06	0.62	166.86	0.3
Avg	168.84	0.62	164.99	0.27

6.2.1. Image reproduction library

GA for reproducing images (GARI) is a Python project using the PyGAD library [52] to reproduce the original input images. GARI is an inbuilt library; hence customization of the algorithmic parameters is not required, unlike other EAs.

6.2.2. GA (through GARI library) for image reproduction

Evolving pixels serve act as a basic principle for image reproduction using a GA. As a first step, an input image is preprocessed to be converted into a 1D (*1-dimensional*) vector. *Initial_population( )* function in the GARI library creates a random population [53]. The fitness function is designed in such a way that it accepts two arguments, representing candidate solutions and their corresponding indices, to return the fitness value. The fitness value is fetched, by accumulation of the sum of the absolute differences between the gene values within the original and reproduced candidates. Candidates are sorted based on their fitness values, and the candidates with the best fitness are selected as parent. Among the (crossover and mutation) variation operations, the *crossover( )* function is invoked for carrying out the crossover operation.

The *crossover( )* function requires three arguments - input image shape (*img_shape*), number of offspring to return (*n_individuals*), and the best performing candidates in the previous generation. The number of individuals (*n_individuals*) is kept as 8 [55]. In the mutation operation, *mutation( ),* some genes in the chromosome are selected at random, and a random weight is added to the gene (similar to the adaptive mutation technique). The *mutation( )* function receives the population returned by the *crossover( )* function. The percentage of change in a chromosome is determined by the best candidate received from the previous generation. Once the variation operations are consummated, *PyGAD.GA* class instance is created. The flowchart of the GARI library workflow is depicted in Fig. 4.

6.2.3. MTDGA for image reproduction

In the MTDGA framework, the master node decomposes an original input image into smaller regions and disseminates them to all the slave nodes. On receipt of the images, the input image is further decomposed into smaller regions, which are sent to the threads that were incorporated within the *GARI* library. The image reproduction process is initiated in the threads, as soon as an input image is received. The further random population is generated, followed by variation operation; on completion of the latter, PyGAD.GA class instance is developed automatically. The spontaneous mutation employed with *mutation_by_replacement* = 'True'. The parameters configured are *range_of_pixel_values*, *init_range_high*, *init_range_low*, *random_mutation_max_val*, and *random_mutation_min_val* in the PyGAD instance [54]. The image pixel value is kept in the range [0 - 255], and *random_mutation_max_val* and *init_range_high* are set to 255. The *random_mutation_min_val* and *init_range_low* are fixed to 0. The execution of the instance is initiated with *run( )* method.

The working of MTDGA is similar to the MTDDE framework; when the thread reaches the maximum number of iterations, the reproduced image slices are sent to respective slave nodes. On receipt of all resultant segments from the threads, the slave nodes recombine them in a specific order, prior to sending them to the master node. The master node, on receipt of all the resultant images from the slaves, produces the final image.

The objective of testing the MTDGA framework for image reproduction is to reproduce an image with quality similar to the original image and minimize the process execution time. A detailed performance analysis was carried out, between the sequential GA and the MTDGA framework, on reproduction of the original image. The performance metrics taken into consideration were – peak signal-to-noise ratio (*PSNR*), structural similarity index metrics (*SSIM*), mean square error (*MSE*), and execution time (*ET*). Table 5 displays the performance of sequential GA for 10,000 generations.

Table 6 depicts the performance of the MTDGA framework for 10,000 generations with horizontal and vertical slicing, for comprehension of their impact on the performance of the MTDGA framework. Results in Table 6 attest that performance of the MTDGA framework is immune to the slicing styles. Comparison of the performances of the MTDGA framework (Table 6) with the sequential GA (Table 5) justify that the quality of the image (in terms of *PSNR* and *SSIM* values) reproduced by the MTDGA framework is much preferred. Besides, MTDGA has less MSE score compared to sequential GA, which reaffirms that image reproduction quality is enhanced using the MTDGA framework. Furthermore, the MTDGA framework has taken only a third of the execution time (*ET*) incurred by the sequential GA model.

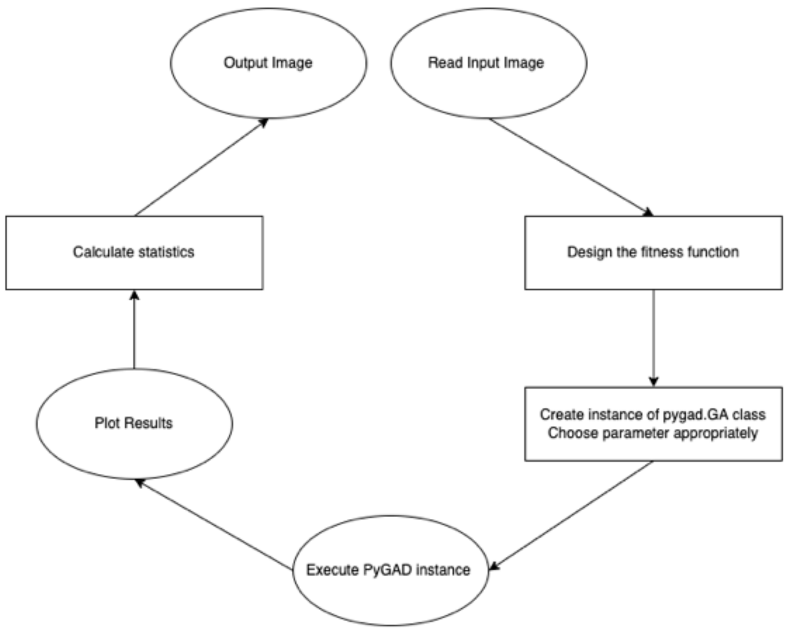
Output Image
Read Input Image
Calculate statistics
Design the fitness function
Plot Results
Create instance of pygad.GA class
Choose parameter appropriately
Execute PyGAD instance
Figure. 4 Flowchart of the GARI library

Table 5. Performance of sequential GA for image reproduction

Run	PSNR	MSE	SSIM	ET (s)
1	7.11	105.64	0.0196	24.80
2	7.11	105.21	0.0211	23.64
3	7.12	105.35	0.0234	23.81
4	7.10	105.46	0.0228	23.72
5	7.14	105.38	0.0206	23.71
6	7.11	105.24	0.0233	23.81
7	7.14	104.87	0.0231	23.97
8	7.11	105.59	0.0208	23.90
9	7.13	105.10	0.0230	23.7 3
10	7.11	104.89	0.0224	23.72
Average	7.12	105.27	0.0220	23.88

Table 7 shows the average comparative performance values for 10 independent runs of these 2 models, when exposed to increased number of generations (20 000, 30 000, 40 000, and 50 000). Table 7 also proves that MTDGA provides a better-reproduced image with minimal execution time, as the number of generations increases. The method of decomposing the image into smaller regions and iterating, until the termination condition is satisfied, is the reason behind increase in quality of the solution afforded by the recommended MTDGA framework. Figure 5 illustrates the evolutionary process of enhanced image reproduction quality with higher numbers of generations.

7. Experimentation of MTDEA on HPC

The MTDEA framework was deployed on a 140 core-HPC system, configured with 1 TB RAM, 21 TB storage and Red Hat Linux (6.10 version) operating system, to explore scalability of the MTDEA framework. The "*eso1242a*" test image was downloaded from European Southern Observatory (ESO) website [56]. This website acts as a repository that contains images of celestial objects with high definition. The size of this image was 1.5 GB (giga-pixels). The performances of MTDDE and MTDGA frameworks were evaluated on the HPC system. The experimental setup followed in previous experiments (Section 6.1) was followed for the current experiment on HPC.

7.1 MTDDE on HPC

Table 8 presents a comparative analysis of the sequential and multi-threaded execution options, employed on the HPC system noted in section 6.1. The *SET* and *DET*, in Table 8, respectively represent sequential and distributed execution times. To provide clear analysis, optimum threshold achieved for each slice of image in MTDDE framework is displayed. AOT refers to average value of the optimal thresholds achieved for the segmented image slices by MTDDE framework. The results advocate that MTDDE framework incorporated in HPC environment is efficient and scalable for image processing applications with larger image size (giga-byte images).

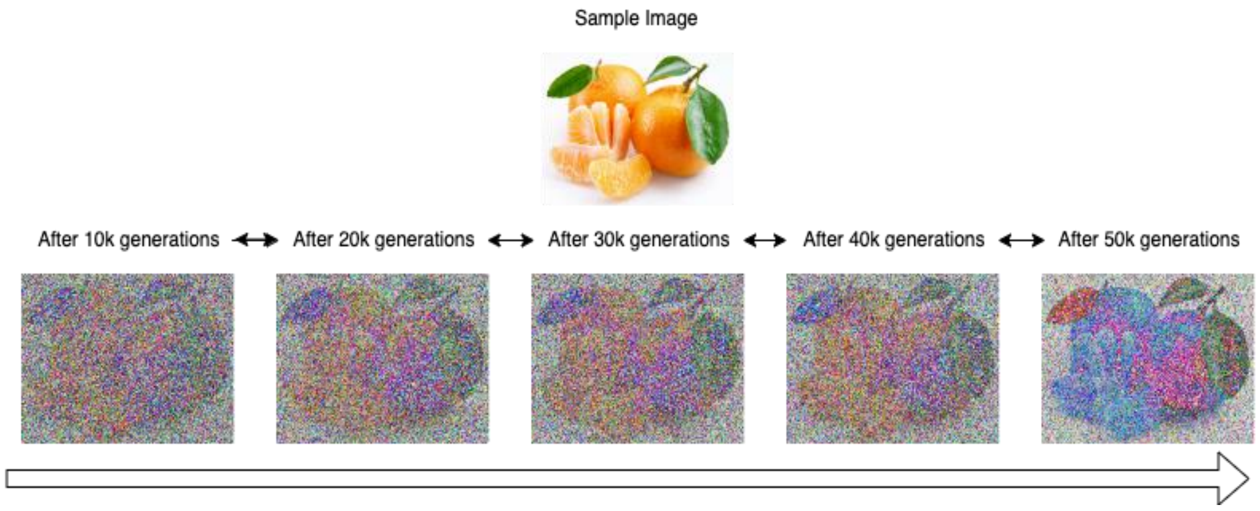

Figure 5. The evolutionary process of image reproduction using MTDGA framework

Table 6. Performance of MTDGA framework for image reproduction

Run	Vertical slicing of image				Horizontal splicing of image			
	PSNR	MSE	SSIM	ET	PSNR	MSE	SSIM	ET
1	7.70	104.76	0.0327	8.75	7.78	104.37	0.0352	8.54
2	7.71	104.60	0.0327	8.78	7.78	104.34	0.0348	8.66
3	7.72	104.90	0.0332	8.79	7.80	104.21	0.0354	8.63
4	7.75	104.79	0.0360	8.73	7.82	104.48	0.0379	8.61
5	7.74	104.41	0.0354	8.77	7.77	104.58	0.0346	8.69
6	7.70	104.79	0.0351	8.73	7.78	104.19	0.0360	8.70
7	7.74	104.98	0.0337	8.69	7.78	105.11	0.0384	8.64
8	7.72	104.80	0.0335	8.89	7.78	104.47	0.0359	8.77
9	7.74	104.67	0.0351	8.77	7.79	104.58	0.0342	8.72
10	7.73	105.04	0.0355	8.85	7.77	104.95	0.0361	8.63
Average	7.72	104.77	0.0343	8.77	7.78	104.53	0.0358	8.66

Table 7. Overall performance comparison of GA vs. MTDGA

Number of Generations	Sequential GA				MTDGA			
	PSNR	MSE	SSIM	ET	PSNR	MSE	SSIM	ET
20 000	7.202	104.873	0.029	34.16	8.403	103.703	0.048	19.20
30 000	7.264	105.110	0.030	50.12	8.574	103.603	0.057	27.25
40 000	7.291	104.651	0.031	68.16	8.636	103.391	0.057	36.78
50 000	7.303	105.576	0.031	83.59	8.675	103.162	0.057	45.62

7.2 MTDGA on HPC

To study the performance of MTDGA on the HPC system, the same test image, but, with a lesser quality (33.6 MB) was chosen. This experiment was aimed to study how well the MTDGA framework performs when exposed to an HPC system. As the MTDGA framework took around 70 hours to process 1.5 GB image and 22 hours to process an image size of 236.2 MB, to simplify the experiment, an image of 33.6 MB size was chosen. Table 9 depicts the performance comparison between sequential GA and MTDGA in the HPC system. The results in Table 9 are an average of 10 runs, to overcome the randomization error.

Table 8. Performance comparison of sequential DE vs. MTDDE in HPC

Runs	Optimal threshold by SDE	SET (s)	Optimal threshold by MTDDE, for each slice of the image			AOT	DET (s)
1	84.16	1931.00	102.06	109.12	56.58	89.26	1098.74
2	83.66	2032.21	101.89	105.63	57.65	88.39	1072.79
3	85.68	2020.60	106.08	106.84	57.87	90.26	1062.18
4	85.37	2067.82	102.42	107.12	58.46	89.33	1081.87
5	82.35	1989.95	101.83	106.88	58.14	88.95	1095.60
6	83.61	2021.50	101.62	105.77	58.26	88.55	1056.71
7	83.98	2007.96	103.21	104.75	56.84	88.26	1051.07
8	84.08	2072.28	101.56	107.45	57.89	88.97	991.09
9	86.45	2089.67	103.74	107.35	57.65	89.58	1007.89
10	83.69	1947.16	102.43	105.86	57.97	88.76	1033.57
AVG	84.30	2018.01				89.03	1055.15

Table 9. Performance comparison of Sequential GA vs. MTDGA on HPC

Number of Generations	Sequential GA				MTDGA			
	PSNR	MSE	SSIM	ET (s)	PSNR	MSE	SSIM	ET (s)
10000	7.92	105.49	0.0071	561.73	7.86	105.40	0.0063	310.22
20000	7.90	105.43	0.0067	735.65	7.81	105.33	0.0058	635.74
30000	7.88	105.46	0.0066	1163.97	7.80	105.32	0.0057	955.22
40000	7.90	105.46	0.0068	1430.56	7.80	105.25	0.0055	1317.61
50000	7.88	105.43	0.0064	2822.56	7.79	105.26	0.0055	1695.30

Table 10. The algorithmic structure of each node in the framework

```
a       For i number of runs
        1       For j number of generations
                a       For k number of candidates
                        1       For d dimensions
                                a       Perform mutation (d steps)
                        2       Perform crossover (assume 1 step)
                        3       Perform selection (assume 1 step)
                b       End of k loop.
                        4       Perform migration (assume 3 steps)
        2       End of j loop
b       End of i loop
```

Results in Tables 8 and 9 affirm that the MTDEA framework performs with minimal execution time without compromising the solution quality on the HPC system. Testing the frameworks on HPC system asserts, that MTDEA is scalable and reliable, in highly distributed computing environment.

8. Time complexity analysis

This section details the time complexity of the MTDEA framework. The parameters used in the MTDEA framework were the maximum number of runs (i), the maximum number of generations (j), the number of candidates (k), and problem dimension (d). The algorithmic structure of a single node in the MTDEA framework is shown in Table 10. In the case of the MTDEA framework (with 3 slave nodes and 2 master node), the estimated time complexity is in the order of $4 * (i * j * (k * (d + 2) + 3))$. The time complexity of the MTDEA framework is in O $(i * j * k * d)$, which is similar to the time complexity of the conventional DE algorithm [57], irrespective of the number of nodes utilized in a parallel fashion.

9. Conclusions

This paper demonstrated the propriety of a fault tolerant multi-threaded distributed evolutionary algorithm framework (MTDEA) to handle benchmarked image processing problems. The major contribution of this study is the performance comparison of image processing applications on MTDEA frameworks with conventional computing and HPC system. The MTDEA framework, stacked against traditional differential evolutionary (DE) models, was shown to be effective, reliable, and exhibit faster process-execution times, devoid of any detriment to image quality. The proffered MTDEA framework, embodying synergistic blend of distributed computing, multi-threading, and evolutionary algorithm, exhibited strong potential to provide promising solutions to different problems. The advocated MTDEA framework was a simulation model that can be conveniently upgraded and customized in a highly distributed computing, as evidenced by its validation on a high-performance computing system.

Despite the credible reassuring performance of the MTDEA framework, the notable limitation of over-segmentation of images needs to be addressed Besides, MTDEA's performance needs to be validated on a larger image dataset. Future research will focus on implementing the MTDEA framework in a highly distributed dynamic computing environment.

Conflicts of interest

No conflicts of interest to the best of our knowledge.

Anthors contributions

This work represents the findings of the scholar(Author-1) during his Ph.D. process and (Author-2) is the mentor of his (Author-1) Ph.D. programme.

References

[1] P. Yang, K. Tang, and X. Yao, "A Parallel Divide-and-Conquer-Based Evolutionary Algorithm for Large-Scale Optimization", *International Journal of IEEE Access*, Vol. 7, pp. 163105-163118, 2019.

[2] Y. Bo, "Image Segmentation of the Genetic Algorithms on the Base of Otsu", *Journal of Natural Science of Hunan Normal University*, pp. 32-36, 2003.

[3] N. Otsu, "A Threshold Selection Method from Gray-Level Histograms", *International Journal of IEEE Transactions on Systems*, Man, and Cybernetics, Vol. 9, No. 1, pp. 62-66, 1979.

[4] J. L. Fan, X. F. Zhang, and Z. Feng, "Three-dimension maximum between-cluster variance image segmentation method based on chaotic optimization", In: *Proc. of International Conf. on Virtual Systems and Multimedia*, 2006.

[5] C. Arif, S. Harsh, S. Hrutuja, J. Dheeraj, and G. Shiwani, "Reproducing Images using Genetic Algorithm", *International Research Journal of Engineering and Technology (IRJET)*, Vol. 8, No. 4, 2021.

[6] S. Raghul and G. Jeyakumar, "A Distributed Multithreaded Evolutionary Computing Frame Work using Differential Evolution Algorithm", In: *Proc. of International Conf. on Inventive Computation Technologies (ICICT)*, pp. 1145-1151, 2021.

[7] S. Raghul and G. Jeyakumar, "Investigations on Distributed Differential Evolution Framework with Fault Tolerance Mechanisms", *In Book: Differential Evolution: From Theory to Practice. Studies in Computational Intelligence*, pp. 175-196, 2022.

[8] R. Storn and K. Price, "Differential evolution—a simple and efficient adaptive scheme for global optimization over continuous spaces", In: *Technical Report-95-012*, ICSI, 1995.

[9] E. M. Montes, J. V. Reyes, and A. Coello, "A comparative study of differential evolution variants for global optimization", In: *Proc. of International Conf. on Genetic and Evolutionary Computation*, pp 485–492, 2006.

[10] K. Price, R. M. Storn, and J. A. Lampinen, "Differential evolution: a practical approach to global optimization", *In Book: Springer Science & Business Media*, 2006.

[11] K. V. Price, "An introduction to differential evolution", *In Book: New ideas in Optimization*, pp. 79–108, 1999.

[12] A. K. Qin, V. L. Huang, & P. N. Suganthan, "Differential evolution algorithm with strategy adaptation for global numerical optimization", *Journal of IEEE transactions on Evolutionary Computation*, Vol. 13, No. 2, pp. 398-417, 2009.

[13] J. H. Holland, "Adaption in Natural and Artificial Systems", *In Book: Control, and Artificial Intelligence*, 1975.

[14] A. P. Bharathi, D. R. Pallavi, M. Ramachandran, K. Ramu, and C. Sivaji, "A Study on Evolutionary Algorithms and Its Applications", *In Book: Electrical and Automation Engineering, REST Publisher*, Vol. 1, No. 1, 2022.

[15] J. Yue, C. W. Neng, Z. Z. Hui, Z. Jun, L. Yun, and Z. Qingfu, "Distributed evolutionary algorithms and their models: A survey of the state-of-the-art", *International Journal of Applied Soft Computing*, Vol. 36, pp. 286-300, 2015.

[16] Y. Zheng, X. Xu, S. Chen and W. Wang, "Distributed agent based cooperative differential evolution: A master-slave model", In: *Proc. of International Conf. on Cloud Computing and Intelligence Systems*, pp. 376-380, 2012.

[17] S. M. Said and M. Nakamura, "Parallel Enhanced Hybrid Evolutionary Algorithm for Continuous Function Optimization", In: *Proc. of International Conf. on P2P, Parallel, Grid, Cloud and Internet Computing*, pp. 125-131, 2012.

[18] O. Abdoun, Y. Moumen and F. Abdoun, "Parallel evolutionary computation to solve combinatorial optimization problem", In: *Proc. of International Conf. on Electrical and Information Technologies (ICEIT)*, pp. 1-6, 2017.

[19] M. Depolli, R. Trobec and B. Filipič, "Asynchronous Master-Slave Parallelization of Differential Evolution for Multi-Objective Optimization", *International Journal of Evolutionary Computation*, Vol. 21, No. 2, pp. 261-291, 2013.

[20] C. Lin, J. Liu, H. Yao, C. Chu and C. Yang, "Performance Evaluation of Parallel Genetic Algorithm Using Single Program Multiple Data Technique", In: *Proc. of International Conf. on Trustworthy Systems and Their Applications*, pp. 135-140, 2015.

[21] A.T. A. Oqaily and G. Shakah, "Solving Non-Linear Optimization Problems Using Parallel Genetic Algorithm", In: *Proc. of International Conf. on Computer Science and Information Technology (CSIT)*, pp. 103-106, 2018.

[22] K. I. Abuzanouneh, "Parallel and Distributed Genetic Algorithm with Multiple-Objectives to Improve and Develop of Evolutionary Algorithm", *International Journal of Advanced Computer Science and Applications*, Vol. 7, No. 5, 2016.

[23] Z. H. Zhan, X. F. Liu, H. Zhang, Z. Yu, J. Weng, Y. Li, T. Gu, and J. Zhang, "Cloudde: A heterogeneous differential evolution algorithm and its distributed cloud version", *Journal of IEEE Transactions on Parallel Distributed Systems*, Vol. 28, No. 3, pp. 704–716, 2017.

[24] R. Panwar and M. Supriya. "Dynamic resource provisioning for service-based cloud applications: A Bayesian learning approach", *International Journal of Parallel and Distributed Computing*, Vol. 168, pp 90-107, 2022.

[25] B. Cao, J. Zhao, Z. Lv and X. Liu, "A Distributed Parallel Cooperative Coevolutionary Multiobjective Evolutionary Algorithm for Large-Scale Optimization", *Journal of IEEE Transactions on Industrial Informatics*, Vol. 13, No. 4, pp. 2030-2038, 2017.

[26] D. N. Harini and R. Karthi, "Performance analysis of genetic algorithm for function optimization in multicore platform using DEAP", In: *Proc. of International Conf. on Soft Computing and Signal Processing*, pp. 269-279, 2022.

[27] Y. Ge, W. Yu, Z. Zhan and J. Zhang, "Competition-Based Distributed Differential Evolution", In: *Proc. of IEEE Congress on Evolutionary Computation (CEC) Conf.*, pp. 1-8, 2018.

[28] P. Yang, K. Tang and X. Yao, "A Parallel Divide-and-Conquer-Based Evolutionary Algorithm for Large-Scale Optimization", *Journal of IEEE Access*, Vol. 7, pp. 163105-163118, 2019.

[29] M. Adamik, J. Goga, J. Pavlovicova, A. Babinec and I. Sekaj, "Fast robotic pencil drawing based on image evolution by means of genetic algorithm", *International Journal of Robotics and Autonomous Systems*, Vol. 148, 2022.

[30] Y. Sun, Y. Li, Y. Yang and H. Yue, "Differential evolution algorithm with population knowledge fusion strategy for image registration", *International Journal of Complex & Intelligent Systems*, Vol. 8, No. 2, pp. 835-850, 2022.

[31] F. F. Liu, S. C. Chu, X. Wang and J. S. Pan, "A collaborative dragonfly algorithm with novel communication strategy and application for multi-thresholding color image segmentation", *Journal of Internet Technology*, Vol. 23, No. 1, pp. 45-62, 2022.

[32] S. R. Meesala and S. Subramanian, "Feature based opinion analysis on social media tweets with association rule mining and multi-objective evolutionary algorithms", *International Journal of Concurrency and Computation: Practice and Experience*, Vol. 34, No. 3, p. 6586, 2022.

[33] J. Luo, F. He, H. Li, X. T. Zeng and Y. Liang, "A novel whale optimization algorithm with filtering disturbance and nonlinear step", *International Journal of Bio-Inspired Computation*, Vol. 20, No. 2, pp. 71-81, 2022.

[34] S. Ray, S. Parai and A. Das "Cuckoo search with differential evolution mutation and Masi entropy for multi-level image segmentation", *International Journal of Multimedia Tools and Applications*, Vol. 81, pp. 4073–4117, 2022.

[35] S. Akshay and S. Deepika, "Categorization of Fruit images using Artificial Bee Colony Algorithm based on GLCM features", In: *Proc. of International Conf. On Electronic Systems and Intelligent Computing (ICESIC)*, pp. 46-51, 2022.

[36] Y. Sun and Y. Yang, "An Adaptive Bi-Mutation-Based Differential Evolution Algorithm for Multi-Threshold Image

Segmentation", *International Journal of Applied Sciences*, Vol. 12, No. 11, p. 5759, 2022.

[37] A. K. Bhandari, "A novel beta differential evolution algorithm-based fast multilevel thresholding for color image segmentation", *International Journal of Neural computing and applications*, Vol. 32, No. 9, 2020.

[38] A. Sharma, R. Chaturvedi, S. Kumar, and U. K. Dwivedi, "Multi-level image thresholding based on Kapur and Tsallis entropy using firefly algorithm", *Journal of Interdisciplinary Mathematics*, Vol. 23, No. 2, pp. 563-571, 2020.

[39] O. Tarkhaneh, and S. Haifeng, "An adaptive differential evolution algorithm to optimal multi-level thresholding for MRI brain image segmentation", *Journal of Expert Systems with Applications*, 2019.

[40] H. Jia, L. Chunbo and O. Diego, "Hybrid grasshopper optimization algorithm and differential evolution for multilevel satellite image segmentation", *International Journal of Remote Sensing*, Vol. 11, No. 9, 2019.

[41] K. D. Gupta, and S. Sajib, "A genetic algorithm approach to regenerate image from a reduce scaled image using bit data count", *Journal of Broad Research in Artificial Intelligence and Neuroscience*, Vol. 9, No. 2, pp. 34-44, 2018.

[42] S. Shivangini and U. Arvind, "Image Enhancement Using Genetic Algorithm", *International Journal of Engineering and Technology (IJERT)*, 2014.

[43] I. Aravind, C. Chandra, M. Guruprasad, P. S. Dev and R. D. Samuel, "Implementation of image segmentation and reconstruction using genetic algorithms", In: *Proc. of International Conf. on Industrial Technology*, pp. 970-975, 2002.

[44] S. Hashemi, S. Kiani, N. Noroozi and M. E. Moghaddam, "An Image Enhancement Method Based on Genetic Algorithm", In: *Proc. of International Conf. on Digital Image Processing*, pp. 167-171, 2009.

[45] J. I. Hidalgo, J. Lanchares, F. F. D. Vega, and Lombrana, "Is the island model fault tolerant?", In: *Proc. of the 9th Annual Conf. Companion on Genetic and Evolutionary Computation*, pp. 2737-2744, 2007.

[46] W. Gropp and E. Lusk, "Fault Tolerance in MPI Programs", In: *Proc. of International Conf. on Cluster Computing and Grid Systems*, 2002.

[47] J. B. M. Litzkow, T. Tannenbaum and M. Livny, "Checkpoint and migration of unix processes in the condor distributed processing system", *Technical Report 1346 at University of Wisconsin Madison Computer Sciences*, 1997.

[48] D. Anderson, "BOINC: a system for public-resource computing and storage", In*: Proc. of Fifth IEEE/ACM International Workshop on Grid Computing*, pp 4-10, 2004.

[49] M. Jelasity, M. Preuss, M. V. Steen and B. Paechter, "Maintaining Connectivity in a Scalable and Robust Distributed Environment", In: *Proc. of the Second IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid2002)*, 2002.

[50] G. Jeyakumar and C. S. Velayutham, "Distributed mixed variant differential evolution algorithms for unconstrained global optimization", *International Journal of Memetic Computing*, Vol. 5, pp. 275–293, 2013.

[51] P. J. Ballester, J. Stephenson, J. N. Carter and K. Gallagher, "Real-parameter optimization performance study on the CEC-2005 benchmark with SPC-PNX", In: *Proc. of IEEE Congress on Evolutionary Computation*, Vol. 1, pp. 498-505, 2005.

[52] X. Yao, Y. Liu, K. H. Liang, and G. G. Lin, "Fast evolutionary algorithms", In: *Proc. of International Conf. on Advances in Evolutionary Computing: Theory and Applications*, pp. 45-94, 2003.

[53] P. Zhenkui, Z. Yanli and L. Zhen, "Image segmentation based on Differential Evolution algorithm", In: *Proc. of International Conf. on Image Analysis and Signal Processing*, pp. 48-51, 2009.

[54] Dana, "Machine learning optimization using genetic algorithm", *Architect and Industrial Engineer: Udemy Instructor*, 2017.

[55] T. Wu, "Image-Guided Rendering with an Evolutionary Algorithm Based on Cloud Model", *International Journal of Computational Intelligence and Neuroscience*, 2018.

[56] European Southern Observatory. Available at: https://www.eso.org/public/

[57] D. M. Dhanalakshmy, G. Jeyakumar, and C. S. Velayutham, "Empirical investigations on evolution strategies to self-adapt the mutation and crossover parameters of differential evolution algorithm", *International Journal of Intelligent Systems Technologies and Applications*, Vol. 20, No. 2, pp. 103-125, 2021.