

## Article

# GPU-Accelerated Infill Criterion for Multi-Objective Efficient Global Optimization Algorithm and Its Applications

Shengguan Xu <sup>1</sup> , Jiale Zhang <sup>1</sup>, Hongquan Chen <sup>1,\*</sup>, Yisheng Gao <sup>1</sup>, Yunkun Gao <sup>2</sup>, Huanqin Gao <sup>1</sup> and Xuesong Jia <sup>1</sup>

<sup>1</sup> Key Laboratory of Non-Steady Aerodynamics and Flow Control of MIIT, College of Aerospace Engineering, Nanjing University of Aeronautics and Astronautics, Nanjing 210016, China

<sup>2</sup> School of Mechanical Engineering, Anhui University of Technology, Maanshan 243002, China

\* Correspondence: hqchenam@nuaa.edu.cn

**Abstract:** In this work, a novel multi-objective efficient global optimization (EGO) algorithm, namely GMOEGO, is presented by proposing an approach of available threads' multi-objective infill criterion. The work applies the outstanding hypervolume-based expected improvement criterion to enhance the Pareto solutions in view of the accuracy and their distribution on the Pareto front, and the values of sophisticated hypervolume improvement (HVI) are technically approximated by counting the Monte Carlo sampling points under the modern GPU (graphics processing unit) architecture. As compared with traditional methods, such as slice-based hypervolume integration, the programming complexity of the present approach is greatly reduced due to such counting-like simple operations. That is, the calculation of the sophisticated HVI, which has proven to be the most time-consuming part with many objectives, can be light in programmed implementation. Meanwhile, the time consumption of massive computing associated with such Monte Carlo-based HVI approximation (MCHVI) is greatly alleviated by parallelizing in the GPU. A set of mathematical function cases and a real engineering airfoil shape optimization problem that appeared in the literature are taken to validate the proposed approach. All the results show that, less time-consuming, up to around 13.734 times the speedup is achieved when appropriate Pareto solutions are captured.

**Keywords:** efficient global optimization; Monte Carlo; hypervolume; GPU; multi-objective optimization



**Citation:** Xu, S.; Zhang, J.; Chen, H.; Gao, Y.; Gao, Y.; Gao, H.; Jia, X. GPU-Accelerated Infill Criterion for Multi-Objective Efficient Global Optimization Algorithm and Its Applications. *Appl. Sci.* **2023**, *13*, 352. <https://doi.org/10.3390/app13010352>

Academic Editor: Wei Huang

Received: 29 November 2022

Revised: 19 December 2022

Accepted: 24 December 2022

Published: 27 December 2022



**Copyright:** © 2022 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

## 1. Introduction

The efficient global optimization (EGO) algorithm, which was first proposed by Jones et al. [1] in 1998, has proved to be very efficient for dealing with expensive-to-evaluate optimizations [2–5]. Benefited from the fundamental concepts of the Kriging surrogate model and the expected improvement (EI) criterion, EGO can always capture a high-quality optimal solution with relatively less objective function evaluations (FEs) compared to other popular optimization algorithms, such as genetic algorithms (GAs) [6]. Because of this advantage, the corresponding theory has been applied to a variety of modern engineering applications [7–11]. One must notice that the original EI criterion is initially designed for single-objective optimization (SOO) problems and cannot be applied to more important multi-objective optimization (MOO) problems directly.

It is well known that real engineering optimization problems usually come up with not only one single objective but also a set of objectives, which usually conflict with each other. Motivated by the outstanding behaviors of the traditional EGO algorithm in dealing with SOO problems, different methods have been developed by modifying the EGO algorithm to be able to cope with MOO problems [12–14], particularly for expensive-to-evaluate problems. For instance, the method, namely ParEGO, was developed by Knowles [12] for coping with MOO problems. Different aggregate single-objectives are updated iteration by iteration using the augmented Tchebycheff functions with different

random vectors to balance the multiple sub-objectives. A similar method called MOEA/D-EGO was later proposed by Liu et al. [14]. These methods, which are realized in a way of transforming the MOO problem into an SOO problem associated with any available single-objective EGO optimizers, do show promise in the test cases appearing in the literature mentioned. However, because of the usage of several probabilistic parameters of corresponding optimization problems, they are usually not easy to directly apply to real time-consuming engineering optimizations [15,16]. In addition, only one optimal solution captured as a result of the transformed SOO problem is mostly weight dependence.

An alternative to avoiding the shortages mentioned is to capture a set of solutions distributed at the Pareto front of the MOO problem. The main challenge in capturing the distribution of Pareto solutions is modifying the EI infill criterion of the EGO algorithm to be suitable for updating Pareto solutions efficiently. For example, Keane et al. [13] developed the Euclidean distance-based EI (EI-ELU) infill criterion by maximizing the minimum distance between the expected central point of probability distribution and the Pareto points. A more recent hypervolume indicator-based criterion, which was proposed by Zitzler and Thiele [17], is more of interest for MOO problems. In recent decades, many practical contributions have been made to hypervolume-based infill metrics, such as works in hypervolume indicator [17,18], hypervolume improvement [19], S-metric [2,20], lower confidence bound (LCB) [2,21], expected hypervolume improvement (EHVI) [18,22], truncated expected hypervolume improvement (TEHVI) [23], and modified expected hypervolume improvement (MEHVI) [24].

In general, the hypervolume-based metric has been reported to be an outstanding metric for improving accuracy and distribution on the Pareto front of the MOO problem [24,25]. However, the calculation of such metrics is still struggling with enhancing the accuracy of computed hypervolume and reducing the computational complexity implementation. In view of accuracy enhancement, several notable methods have been successfully developed for computing hypervolume, such as the LebMeasure method [21,26], Overmars, and Yap method [27], hypervolume by slicing objectives (HSO) method [28], IHSO method [29], HypE method [30], and box decomposition method [31]. These methods have shown their powerful potential in obtaining a more accurate hypervolume value; however, their implementations are usually hugely complex due to the irregular geometry of the nondominated region of the Pareto front, particularly for the unimagined Pareto front caused by many objectives. Alternatively, a Monte Carlo-based method is proposed by Emmerich [2,18] to reduce the computational complexity of related implementation, but high computational costs are still required due to the fact that there are many repeated nondominated calculations related to the Monte Carlo-based calculations of hypervolume. It can be imagined that such situations will be even worse for many objectives involved in the MOO problem. This might be the reason that the Monte Carlo-based method has rarely been reported in real applications of MOO problems in recent years. In contrast, for calculating hypervolume involved in infill criteria, the Monte Carlo-based method, a kind of statistical method, is quite easy to program in implementation, which is more preferred for MOO applications by new users. Therefore, despite the low efficiency associated with calculations of hypervolume, the attractive programming simplicity of the Monte Carlo-based method still motivates the present research to find an efficient way to propose new treatments.

In the present work, an effort has been made to develop a novel approach for computing multi-objective infill criteria for efficient global optimization (EGO) algorithms. The work applies the outstanding hypervolume-based expected improvement metric to enhance Pareto solutions in view of their accuracies and distributions on the Pareto front, and the values of sophisticated hypervolume improvement (HVI) are technically approximated by counting the Monte Carlo sampling points under the modern GPU (graphics processing unit) architecture. The calculation of the sophisticated HVI, which has proven to be the most time-consuming part with many objectives, can be light in programmed implementation due to such counting-like simple operations. Meanwhile, the time consuming of massive computing associated with such HVI approximation is greatly alleviated by parallelizing

in the GPU. It can be learned from the test cases that significant speedups are achieved by the proposed GMOEGO algorithm.

The rest of the paper is organized as follows. The related works, including an overview of the traditional EGO algorithm and the hypervolume-based EI criterion, are described in Section 2. Then, in Section 3, the modified EGO-based multi-objective optimization method is presented by adopting the new proposed GPU-based Monte Carlo method. After presenting representative function tests in Section 4, a more practicable aerodynamic optimization case is further presented to investigate the efficiency of the proposed algorithm in Section 5. Finally, in Section 6, concluding remarks are drawn.

## 2. Related Works

### 2.1. Traditional Single-Objective EGO Algorithm

A study of EGO to be used for optimizations with multiple objectives. The traditional EGO algorithm is usually developed for optimization with a single objective. For the sake of completeness, a brief discussion of the traditional single-objective EGO algorithm [1] is presented in this section. To be the core components of the traditional EGO algorithm, the Kriging surrogate model and EI infill criterion ensure its high efficiency of EGO and global searching ability.

Kriging was first proposed by Krige [32] in 1951, and its formulation with  $d$  variables and  $n_s$  samples can be described as follows: let  $n_s$  sampled points be  $\mathbf{X}^s = \{\mathbf{x}^1, \mathbf{x}^2, \dots, \mathbf{x}^{n_s}\}$ , and their associated objective function values are  $\mathbf{Y}^s = \{\mathbf{y}^1, \mathbf{y}^2, \dots, \mathbf{y}^{n_s}\}$ , in which the  $i$ -th sample point and its objective value are  $\mathbf{x}^i = (x_1^i, x_2^i, \dots, x_d^i)$ ,  $\mathbf{y}^i = (y_1^i, y_2^i, \dots, y_d^i)$ , respectively, for  $i = 1, 2, \dots, n_s$ .

$$Y(\mathbf{x}^i) = \sum_{j=1}^k \beta_j f_j(\mathbf{x}^i) + z(\mathbf{x}^i) \tag{1}$$

where  $f_j(\mathbf{x}^i)$  are  $k$  known regression models, and  $\beta_j$  are correlation coefficients of  $f_j(\mathbf{x}^i)$ ,  $z(\mathbf{x}^i)$  is a stochastic model and its mean is zero and variance is  $\delta_z^2$ . The covariance between two design points  $\mathbf{x}^i$  and  $\mathbf{x}^j$  can be written as

$$Cov[z(\mathbf{x}^i), z(\mathbf{x}^j)] = \delta_z^2 \mathbf{R}[R(\mathbf{x}^i, \mathbf{x}^j)] \tag{2}$$

where  $i, j = 1, 2, \dots, n_s$ ; a symmetric matrix,  $\mathbf{R}$ , is a correlation function that reflects the relationship between all the sample points;  $R$  is a correlation function defined by the user, and a widely used form of  $R$  can be expressed as [33]

$$R(\mathbf{x}^i, \mathbf{x}^j) = \prod_{k=1}^d \exp(-\theta_k |x_k^i - x_k^j|^{p_k}) \tag{3}$$

where  $\theta_k > 0, 0 \leq p_k \leq 2$  are two hyper parameters. Once the hyper parameters  $\theta_k$  and  $p_k$  are determined, the prediction  $\hat{y}(\mathbf{x})$  of  $Y$  at the location  $\mathbf{x}$  can be calculated by

$$\hat{y}(\mathbf{x}) = \hat{\beta} + \mathbf{r}^T(\mathbf{x})\mathbf{R}^{-1}(\mathbf{y} - \mathbf{1}\beta) \tag{4}$$

in which the  $n$ -vector,  $\mathbf{r}(\mathbf{x})$ , reflects the correlations between the predicted location  $\mathbf{x}$  and the sampled locations  $R(\mathbf{x}, \mathbf{x}^j)$ , where  $j = 1, 2, \dots, n_s$ ;  $\mathbf{1}$  is an  $n$ -vector of ones; and an  $n$ -vector,  $\mathbf{y}$ , is the value of accurate response;  $\beta$  is the mean of  $Y(\mathbf{x})$ , and  $\hat{\beta}$  denotes the generalized least squares estimator of  $\beta$ , which is defined as  $\hat{\beta} = (\mathbf{1}^T \mathbf{R}^{-1} \mathbf{1})^{-1} \mathbf{1}^T \mathbf{R}^{-1} \mathbf{y}$  [33], and then, the mean squared error (MSE) of  $\hat{y}(\mathbf{x})$  can be written as

$$MSE[\hat{y}(\mathbf{x})] = \delta_z^2 \left(1 - \mathbf{r}^T(\mathbf{x})\mathbf{R}^{-1}\mathbf{r}(\mathbf{x})\right) \tag{5}$$

The hyper parameter  $\theta_k$  (see Equation (3)) can be determined by using the maximum likelihood estimation (MLE) approach with the limitation of  $\theta_k > 0$

$$\text{Ln(Likelihood)} = -\frac{1}{2} \left( n \ln(\delta_z^2) + \ln|R| \right) \tag{6}$$

where  $\delta_z^2 = (\mathbf{y} - \mathbf{1}\hat{\beta})^T \mathbf{R}^{-1} (\mathbf{y} - \mathbf{1}\hat{\beta}) / n$ , and by solving Equation (6), the Kriging model can be constructed. The approximation  $\hat{y}(\mathbf{x})$  and MSE can be obtained by Equations (4) and (5).

Once a Kriging surrogate model is constructed, the EI function (Equation (7)) can be determined using a specific optimizer. Assume the  $\hat{y}(\mathbf{x})$  follows the normal distribution  $Y \sim N(\mu, s^2)$ , where  $\mu$  is the Kriging predictor  $\hat{y}(\mathbf{x})$  defined in Equation (4),  $s^2(\mathbf{x})$  is the MSE (see Equation (5)), and  $y_{\min}$  is the minimum of all the sample points. Then the EI function at location  $\mathbf{x}$  can be expressed as follows [1]

$$E[I(\mathbf{x})] = \begin{cases} (y_{\min} - \hat{y}(\mathbf{x}))\Phi\left(\frac{y_{\min} - \hat{y}(\mathbf{x})}{s(\mathbf{x})}\right) + s(\mathbf{x})\phi\left(\frac{y_{\min} - \hat{y}(\mathbf{x})}{s(\mathbf{x})}\right) & s(\mathbf{x}) > 0 \\ 0 & s(\mathbf{x}) = 0 \end{cases} \tag{7}$$

where  $\Phi$  and  $\phi$  are the normal cumulative distribution function and the normal probability density function, respectively. This criterion is considered a balance of “exploration and exploitation”.

It should be pointed out that the EI criterion is initially designed for SOO problems, and cannot be directly applied to MOO problems. That is, for MOO problems, the new EI criterion should be constructed for the multi-objective EGO algorithm, which will be discussed in the next section.

### 2.2. EHVI Infill Criterion for the Multi-Objective EGO Algorithm

As discussed in Section 1, one of the most popular criteria designed for a multi-objective EGO (MOEGO) algorithm is the EHVI criterion, which is modified from HVI. Therefore, the HVI is described here before presenting the EHVI criterion.

Considering the MOO problem with  $m > 1$  objective functions  $f_m: \mathcal{X} \mapsto \mathbb{R}^m$ , the MOO problem can be briefly expressed as

$$\left. \begin{aligned} \text{Minimize}_{\mathbf{x} \in \mathcal{X}} &= \{f_1(\mathbf{x}), f_2(\mathbf{x}), \dots, f_m(\mathbf{x})\} \\ \text{Subject to } &\mathbf{x} \in [l, \mathbf{u}] \end{aligned} \right\} \tag{8}$$

where  $\mathcal{X} \subseteq \mathbb{R}^d$ ,  $\mathbf{y} \in \mathbf{Y} \subseteq \mathbb{R}^m$ .  $l$  and  $\mathbf{u}$  are the lower and upper constraints, respectively, of the design variable vector  $\mathbf{x}$ .  $d$  denotes the number of design variables, and  $m$  is the number of objectives.

Assume the problem (8) to be optimized iteration by iteration with the EGO optimizer and take the set of Pareto front solutions (points) at the  $n$ -th iteration of problem (8) as  $\mathcal{P}_n = \{\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_{n_{dom}}\}$ , where  $n_{dom}$  is the number of current Pareto front points; all the elements in  $\mathcal{P}_n$  are nondominated by each other, and can be mathematically expressed as

$$\forall i, j \in \{1, 2, \dots, n_{dom}\} \wedge (i \neq j): (\mathbf{y}_i \not\prec \mathbf{y}_j) \wedge (\mathbf{y}_j \not\prec \mathbf{y}_i)$$

in which  $a \not\prec b$  means  $a$  does not dominate  $b$ , and on the contrary,  $a \prec b$  means  $a$  dominates  $b$ .  $\mathbf{y}_i = \{y_i^1, y_i^2, \dots, y_i^m\}$  denotes the coordinate of the  $i$ -th point in  $\mathcal{P}_n$ .

In order to compute the hypervolume used in the EHVI criterion, a reference point  $\mathcal{R} = \{r_1, r_2, \dots, r_m\} \in \mathbb{R}^m$  is usually required to be defined by designers before the calculation, and then the hypervolume of the current Pareto front related to the reference point,  $\mathcal{H}(\mathcal{P}_n, \mathcal{R})$ , can be expressed as

$$\mathcal{H}(\mathcal{P}_n, \mathcal{R}) = \int_{(\exists \mathbf{y}_i \in \mathcal{P}_n) \prec \mathbf{y} \prec \mathcal{R}} d\mathbf{y} \tag{9}$$

In Equation (9), one must notice that the hypervolume is equivalent to the region, which is dominated by at least one point of  $\mathcal{P}_n$  and itself dominates the reference point, as shown in Figure 1, shaded with blue oblique lines.

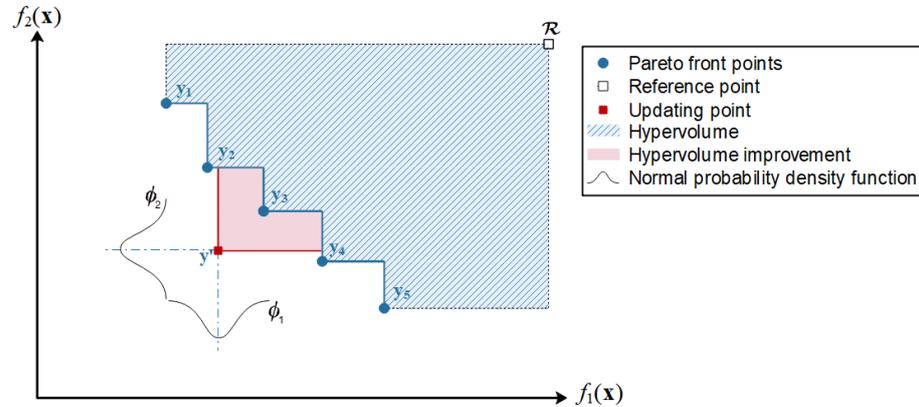


Figure 1. Example of a hypervolume indicator with two objectives.

Assume that the current predicted updating point is  $\mathbf{y}'$ , the HVI indicator  $I_{\mathcal{H}}(\mathbf{y}', \mathcal{P}_n)$  of the red area in Figure 1 can then be defined as:

$$I_{\mathcal{H}}(\mathbf{y}', \mathcal{P}_n) = \begin{cases} \mathcal{H}((\mathcal{P}_n \cup \{\mathbf{y}'\}), \mathcal{R}) - \mathcal{H}(\mathcal{P}_n, \mathcal{R}) & \nexists \mathbf{y}_i \in \mathcal{P}_n : \mathbf{y}_i \prec \mathbf{y}' \\ 0 & \text{Otherwise} \end{cases} \quad (10)$$

It can be noted that only when  $\mathbf{y}'$  falls in the  $\mathcal{P}_n$  nondominated region can the positive value of HVI be obtained. As expected, a large value of HVI reflects an improvement in the Pareto front solutions. Hence, the HVI,  $I_{\mathcal{H}}(\mathbf{y}', \mathcal{P}_n)$ , can be taken as the infill criterion directly, and its further enhanced criterion of the HVI indicator, namely  $EI_{\mathcal{H}}(\mathbf{x})$  of EHVI indicator, is proposed by Emmerich et al. [2] and could be predicted by the expected improvement function  $E[I_{\mathcal{H}}]$ , which reads

$$EI_{\mathcal{H}}(\mathbf{x}) = E[I_{\mathcal{H}}] = \int_{\mathbf{y}' \in \Omega} I_{\mathcal{H}}(\mathbf{y}', \mathcal{P}_n) \text{PDF}(\mathbf{y}') d\mathbf{y}' \quad (11)$$

where the nondominated region  $\Omega$  of the current Pareto front is defined as

$$\Omega = \{ \mathbf{y} \in \mathbb{R}^m \mid (\mathbf{y} \prec \mathcal{R}) \wedge (\nexists \mathbf{y}_i \in \mathcal{P}_n : \mathbf{y}_i \prec \mathbf{y}) \} \quad (12)$$

PDF( $\mathbf{y}'$ ) is the probability density function, which can be estimated by

$$\text{PDF}(\mathbf{y}'(\mathbf{x})) = \begin{cases} \prod_{i=1}^m \phi\left(\frac{y'_i - \hat{y}_i(\mathbf{x})}{s_i(\mathbf{x})}\right) & s_i(\mathbf{x}) > 0 \\ 1 & s_i(\mathbf{x}) = 0 \end{cases} \quad (13)$$

It can be learned that the enhancement of the EHVI criterion is realized by considering the uncertainty of the prediction with the probability density function (see  $\phi_1$  and  $\phi_2$  in Figure 1), which could balance the exploration and exploitation. Unfortunately, there is a lack of direct calculating formula of both the HVI and EHVI criteria due to the fact that the shape of the Pareto front usually appears to be extremely irregular, as illustrated in Figure 2. This results in the calculation of the HVI or EHVI criteria appearing to be of current interest and usually requiring sophisticated programming steps with high time-consuming for both implementation and computation [30,34]. In order to have an improvement, an alternative Monte Carlo-based approach with GPU acceleration is proposed in the present work for computing these infill criteria, which will be mainly addressed in the next section.

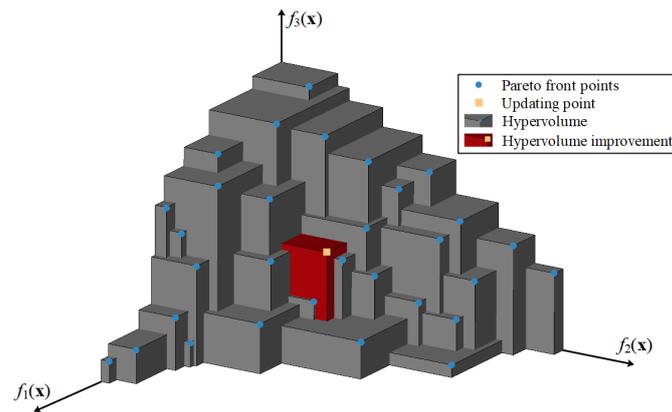


Figure 2. Example of geometry shape of hypervolume and its improvement.

### 3. Novel Approach to Computing Infill Criteria for MOEGO

A novel approach to calculating the hypervolume-based infill criterion for the MOEGO algorithm is developed here by coupling with the Monte Carlo approach. In order to make it clear, in Section 3.1, the main concepts of the Monte Carlo approach are briefly described. In Section 3.2, the GPU-accelerated infill criterion is constructed for the MOEGO algorithm, and in Section 3.3, the resulting framework of the MOEGO algorithm is discussed theoretically.

#### 3.1. A Brief Description of the Monte Carlo Approach

For the sake of easy implementation, the Monte Carlo approach is considered to calculate the HVI or EHVI in the present work. According to the principle of the theory [35], the HVI (see Figure 1 and Equation (10)) could be calculated via the hit-or-miss method (also called the rejection method) [36]. The basic structure of the Monte Carlo-based HVI (MCHVI) is presented for obtaining the HVI, as shown in Algorithm 1.

---

#### Algorithm 1. MCHVI algorithm

---

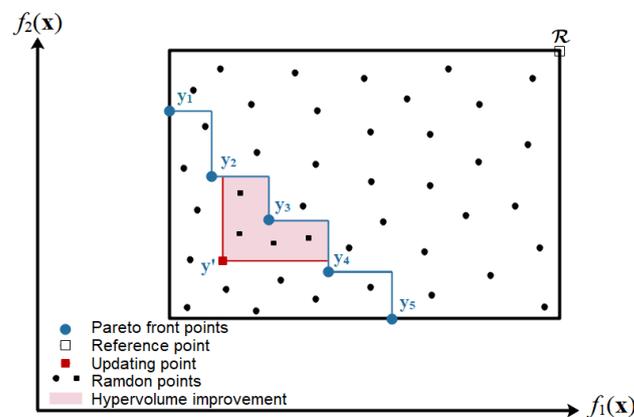
- Input:** Current set of Pareto front points  $\mathcal{P}_n$ , Predicted updating point  $\mathbf{y}'$ , reference point  $\mathcal{R}$ .
- Output:** Hypervolume improvement  $I_{\mathcal{H}}(\mathbf{y}', \mathcal{P}_n)$
- 1 Initialize the Monte Carlo sampling hypercube:
  - 2  $[\min(y_1^i, y_2^i, \dots, y_n^i, r_i), \max(y_1^i, y_2^i, \dots, y_n^i, r_i)]$ ,  $i = 1, 2, \dots, m$ ;
  - 3 Generate a set of  $n_p$  sample points:  $\mathbf{P} = \{p_i(p_i^1, p_i^2, \dots, p_i^m) \in \mathbb{R}^m \mid i = 1, 2, \dots, n_p\}$ ;
  - 4 Initialize count indicators  $k$  and  $n_q$ :  $k = 1$ ,  $n_q = 0$ ;
  - 5 **While**  $k \leq n_p$  **do**
  - 6     **If**  $p_k$  satisfy:  $(\nexists y_i \in \mathcal{P}_n: y_i < p_k) \wedge (\mathbf{y}' < p_k)$  **then**
  - 7         Update  $n_q$ :  $n_q = n_q + 1$ ;
  - 8     Update  $k$ :  $k = k + 1$ ;
  - 9 Compute the volume of hypercube  $V$ :  $V = \prod_{i=1}^m |\max(y_1^i, y_2^i, \dots, y_n^i, r_i) - \min(y_1^i, y_2^i, \dots, y_n^i, r_i)|$ ;
  - 10 Compute and return the estimated MCHVI:  $I_{\mathcal{H}}(\mathbf{y}', \mathcal{P}_n) \approx \frac{n_q}{n_p} \times V$ .
- 

In order to make it clear, Monte Carlo sampling points and approximation of the MCHVI indicator are illustrated with two objectives as shown in Figure 3. In the present

work, the value of the MCHVI indicator is approximated by counting the number  $n_q$  of the sampling points located in the domain of dependence (red area in Figure 3) by

$$I_{\mathcal{H}}(\mathbf{y}', \mathcal{P}_n) \approx \frac{n_q}{n_p} \times V \tag{14}$$

where  $n_p$  is the number of Monte Carlo sampling points (see line 3 of Algorithm 1), and  $V$  is the volume of the hypercube. Such counting-like simple operations are light in programmed implementation, and it can be imagined that the value of the MCHVI indicator is proportional to the size of the Monte Carlo sampling points. However, in order to avoid misguidance caused by the approximate MCHVI indicator, the number of Monte Carlo sampling points is preferred to be as large as possible, which in turn requires massive arithmetic operations associated with predicting the value of the MCHVI indicator will be required. This situation could be alleviated through GPU accelerating, which will be mainly considered and addressed in Section 3.2.



**Figure 3.** Example of HVI calculation by the Monte Carlo-based method with two objectives.

Once the value of HVI indicator  $I_{\mathcal{H}}$  is obtained, the value of EHVI indicator  $EI_{\mathcal{H}}$  can be approximated by the generalized form of the Monte Carlo estimator [37], which can be expressed as

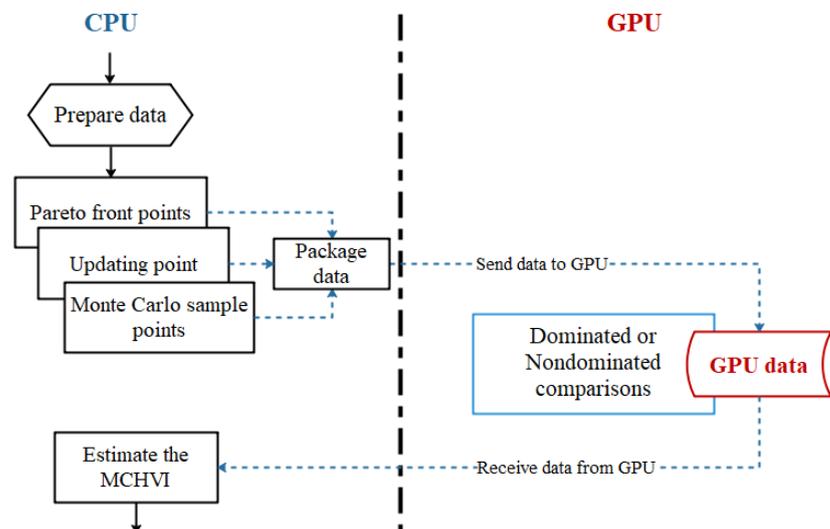
$$EI_{\mathcal{H}}(\mathbf{x}) \approx \frac{1}{N} \sum_{i=1}^N I_{\mathcal{H}}(\mathbf{y}^{mc}_i, \mathcal{P}_n) \tag{15}$$

where  $\{\mathbf{y}_1^{mc}, \mathbf{y}_2^{mc}, \dots, \mathbf{y}_N^{mc}\}$  are the  $N$  random samples of the Gaussian distribution with mean  $\hat{\mu}$  and standard deviation  $s$ . The number  $N$  is a user-defined parameter, and in this paper, following the work of [38], a fixed number of 1000 is used for all test cases presented in Section 4. The  $I_{\mathcal{H}}(\mathbf{y}^{mc}_i, \mathcal{P}_n)$  is predicted by the MCHVI algorithm (see Algorithm 1), which is the main time-consuming part of the EHVI indicator. and hence will be implemented in GPU architecture, as discussed in the next section.

### 3.2. GPU-Accelerated Infill Criterion for the MOEGO Algorithm

In this section, GPU implementation is presented for computing the value of the MCHVI indicator. As described in Algorithm 1, a mass of independent arithmetic operations associated with the dominated or non-dominated identifications (see Algorithm 1, line 6) are proved to be time-consuming. Fortunately, such tasks are mostly weak-dependent compute-intensive and are very suitable for GPU parallel architecture [39–41]. Therefore, such a kind of computation is implemented on the GPU to achieve acceleration. In order to obtain a global view of our implementation, the corresponding general framework of the program is given in Figure 4. It should be noted that necessary computing data, including the coordinates of all Monte Carlo sample points, Pareto front points, and updating points, are designed to be prepared on the CPU side and then sent to the GPU side before starting

the GPU calculations. After all the calculations of the GPU side are finished, the total number of sample points that satisfy the definition of the MCHVI indicator is fetched (see Algorithm 1, line 6), and then sent back to the CPU side for the MCHVI estimation. That is, only time-consuming and suitable tasks are implemented on the GPU.



**Figure 4.** General framework of GPU-based MCHVI indicator value calculations.

Algorithm 2 presents the key code snippet of the GPU subroutine for MCHVI calculations, in which  $MCSP\_d$  is the number of Monte Carlo sample points,  $Front\_d$  denotes the Pareto front, and  $HV\_d$  is the computed MCHVI number  $nq$  (see lines 7 and 8 in Algorithm 1). The corresponding thread hierarchy is designed as shown in Figure 5. It can be learned that a GPU thread is assigned to the calculations related to each Monte Carlo sample point. All GPU threads are organized into a queue of thread blocks inside a thread grid to be adjusted to the double-layer structure of GPU hardware [42]. In the corresponding code snippet (Algorithm 2), the thread index is first calculated using three build-in parameters,  $blockdim$ ,  $blockIdx$  and  $threadIdx$  (for details, see [42]). After the thread index is obtained, the locations of each Monte Carlo sample point (with the same thread index) are then compared with the existing Pareto front and current updating points. If the Monte Carlo point is located in the dominated region, an integer of 1 is added to the value of the MCHVI indicator using a build-in function of *atomicadd* (see line 8 in Algorithm 2).

---

**Algorithm 2.** Code snippet of the kernel for GPU-based MCHVI calculation

---

```

1  attributes(global) subroutine kernel_MCHV(MCSP_d, Front_d, HV_d)
2     $i = (blockIdx \% x - 1) * blockDim \% x + threadIdx \% x$  !thread Index
3
4    do  $j = 1, nFront$  !loop over all Pareto front points and updating point
5      if(isInvalid(MCSP_d(:,i), Front_d(:,j))) return !judge of domination
6    end do
7
8     $istat = atomicadd(HV\_d, 1)$  !accumulate result to global memory
9  end subroutine

```

---

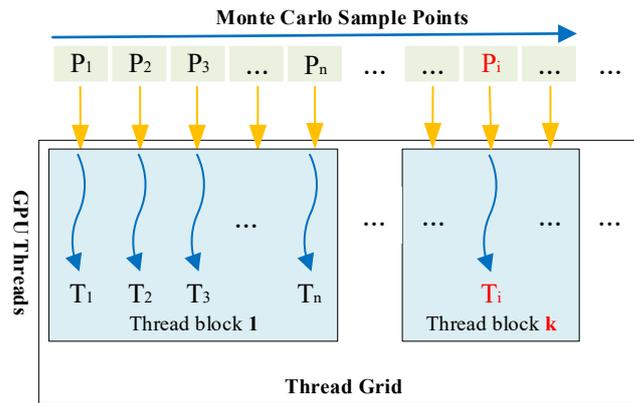


Figure 5. Thread hierarchy designed for GPU-based MCHVI indicator.

While the corresponding CPU subroutine for calling the GPU subroutine (Algorithm 2) is presented in Algorithm 3, in which MCSP is the number of Monte Carlo sample points, Front de-notes the Pareto front, and HV is the computed MCHVI number  $n_q$  (See lines 7 and 8 in Algorithm 1).in which the parameters of thread hierarchy, including number of threads per thread block and total number of thread blocks, are assigned in a specific build-in symbol of “<<<.>>>”. Following our previous GPU works [40,41], a suggested value of 64 is assigned as the number of threads per block; hence, the total number of thread blocks is equal to  $\text{ceiling}(n_p, 64)$ , a rounding function to return the least integer greater than or equal to its argument, in which  $n_p$  is the total number of Monte Carlo sample points. The efficiency of the MOO problem could be expected to improve due to the GPU accelerated implementation, which will be addressed in the following sections.

**Algorithm 3.** Code snippet of the CPU subroutine for calling *kernel\_MCHV*

```

1  subroutine calKernel_MCHV(MCSP, Front, HV)
2    NTPB = 64  !Number of threads per block
3    NBPG = ceiling(MCSP/64)  ! Number of blocks per grid
4
5    MCSP_d = MCSP; Front_d = Front  !Copy data to GPU
7
8    call kernel_MCHV <<<NBPG,NTPB>>> (MCSP_d,Front_d,HV_d) !call the kernel
9
10   HV = HV_d !Send result back to CPU
11  end subroutine

```

3.3. Multi-Objective EGO Method with Modified Infill Criterion

The resulting GPU accelerated Monte Carlo-based multi-objective EGO algorithm, namely GMOEGO, can be summarized as Algorithm 4, as follows.

In the present work, we use the Latin Hypercube Sampling (LHS) [43] is selected as the DOE method (see step 1) for its space filling properties. A global genetic algorithm (GA) is selected to be used in constructing Kriging surrogate models (see step 2) and searching the optimal updating location (in step 4) for the objectives of them are both functions (see Equations (6) and (15)), which are easy for GA optimizers; for details about GA, we refer to [6]. The stopping criterion in step 6 is limited by the maximum number of FEs.

The algorithm presented has been programed to be different modules with Fortran language. The performance of the algorithm presented will be demonstrated and analyzed through comparison with the traditional EGO algorithm, which will be mainly addressed in the next section.

Algorithm 4. GMOEGO Algorithm	
Step 1	<b>Initialization:</b> Use the DOE method to generate a set of sample points $Xs^{n_s}$ within design space, and evaluate their objective values $Ys^{n_s}$ of $Xs^{n_s}$ (for definition of $n_s$ , see Section 2.1).
Step 2	<b>Updating Model:</b> Construct $m$ Kriging surrogate models based on current sample points $(Xs^{n_s}, Ys^{n_s})$ for each objective.
Step 3	<b>Nondominated sort:</b> Sort the current sample points $(Xs^{n_s}, Ys^{n_s})$ by nondominated strategy to obtain the current set of Pareto front point $\mathcal{P}_n$ .
Step 4	<b>Searching for Optimal Updating Point:</b> Based on the Kriging models constructed, search for the optimal updating location by maximizing the EHVI indicator (Equation (11)) based on the Kriging models constructed, in which the indicator is calculated based on the Monte Carlo approach on a GPU computational platform.
Step 5	<b>Objective Function Evaluation:</b> Calculate the values of objective functions at the optimal updating location obtained in Step 4 to update the sample values to obtain $(Xs^{n_s+1}, Ys^{n_s+1})$ .
Step 6	<b>Stopping criterion:</b> Check the stopping criterion. If satisfied, output the optimized Pareto front points and stop; if not, go back to Step 2.

#### 4. Numerical Tests and Analysis

In order to have a view of performance, seven representative mathematical functions with different properties appeared in the literature [16,38,44] are carefully selected to validate the present algorithm. For the sake of convenience, the definitions of the selected functions are listed in the Appendix A.

All the tests are carried out on a personal computer with the Intel Core i9-9900k CPU (8 cores) and GTX-1066 GPU. The code is developed in CUDA Fortran, and the operating system is Windows 10. The first function (ZDT1) is used in Section 4.1 to investigate the speedup effect of the number of Monte Carlo sample points. In Section 4.2, all test functions are then used to validate the present GMOEGO algorithm in comparison with the optimizers listed in Table 1. In the following sections, the size of the initial EGO sample points is set as  $n_s = 6 \times d - 1$ , and all optimizations are carried out under the limited maximum number of FEs. In the present work,  $n_{FEs} \leq 200$  is assigned to be the stop criterion used in step 6 of Algorithm 4. The speedup ratio  $SUR$  of the total computation cost between the two algorithms is defined as

$$SUR = \frac{T_{GMOEGO}}{T_{MOEGO}} \tag{16}$$

where  $T_{MOEGO}$  is the total computational cost of the reference MOEGO algorithm, which is parallelized on the platform of Intel Core i9-9900k CPU with 8 cores, and  $T_{GMOEGO}$  is the total computational cost of the present GMOEGO algorithm, which is run on the GTX-1066 GPU platform.

**Table 1.** Test results of the speedup ratio of ZDT1.

Number of Monte Carlo Sample Points	Algorithm	Total Cost (Hours)	MCHVI Cost (Hours)	SUR of Total Cost (Times)
$1 \times 10^2$	MOEGO	0.239	0.001	0.916
	GMOEGO	0.261	0.018	
$1 \times 10^3$	MOEGO	0.253	0.010	0.920
	GMOEGO	0.275	0.032	
$1 \times 10^4$	MOEGO	0.403	0.169	1.404
	GMOEGO	0.287	0.051	
$1 \times 10^5$	MOEGO	2.146	1.893	5.896
	GMOEGO	0.364	0.124	
$1 \times 10^6$	MOEGO	28.081	27.830	12.575
	GMOEGO	2.233	1.986	

In order to assess the performance of the GMOEGO algorithm, the Inverted Generational Distance (IGD) metric [45] is used in the numerical tests in this section. The IGD metric can be mathematically expressed as

$$IGD(P^*, P) = \frac{\sum_{v \in P^*} d(v, P)}{|P^*|} \tag{17}$$

in which  $P^*$  is usually a set of  $n$  points uniformly distributed on the theoretical Pareto front, while  $P$  is the optimized Pareto front.  $d(v, P)$  is the minimum Euclidean distance between  $v$  and the points in  $P$ , and  $|P^*|$  represents the number of points in  $P^*$ . It can be seen that a low value of  $IGD(P^*, P)$ , means the optimized  $P$  is very close to the theoretical Pareto front  $P^*$ . This is desired by designers.

#### 4.1. Analysis of GPU Speedup Effect

As mentioned above, the computational cost of the MCHVI calculation on the CPU is very time consuming (the most consuming part). In order to investigate the speedup effect of the number of Monte Carlo sample points, the GMOEGO algorithm with different amounts of Monte Carlo sample points,  $1 \times 10^2$ ,  $1 \times 10^3$ ,  $1 \times 10^4$ ,  $1 \times 10^5$ , and  $1 \times 10^6$ , is validated, and the traditional MOEGO algorithm is taken as the reference optimizer for comparison. For the purpose of saving testing time, one test function, the ZDT1 function with 2 variables ( $d = 2$ ), is selected for analyzing the performance of GPU implementation.

The test results for the speedup ratio of ZDT1 are listed in Table 1. It can be learned that the most time-consuming part, up to 99 percent of the overall costs (the MOEGO algorithm with  $1 \times 10^6$  Monte Carlo sample points), are really spent in MCHVI calculations (see the fourth column of Table 1), the time consuming of GMOEGO presented are now speedup by proposed GPU implementation, and significant acceleration, up to 12.575 speedup ratio, are achieved (see the last column in Table 1). In order to make it clear, the corresponding speedup ratio of the total cost to a different number of Monte Carlo sample points is illustrated in Figure 6. When the Monte Carlo sample points are few, it fails to have GPU speedup (see Figure 6) due to the fact that the cost of data exchange between the CPU and GPU side is mainly dominated in such circumstances, as listed in Table 1. It can also be observed that a continual increase in speedups related to the growing number of sample points can be achieved, which are favorable for coping with the problems in the case of a large number of sample points required. Such situations usually occur in complex MOO problems under the requirement of high accuracy in capturing the Pareto front.

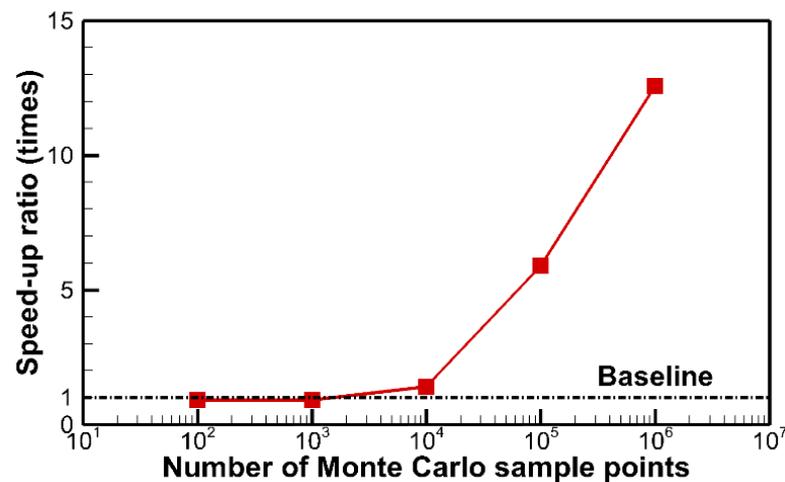


Figure 6. Speedup ratio of total cost to different numbers of Monte Carlo sample points.

#### 4.2. Numerical Tests of the GMOEGO Algorithm

In order to investigate the performance of the proposed GMOEGO algorithm, all six test functions (see Appendix A) are selected as the test functions. In order to mimic real engineering optimizations, test functions with more variables are constructed: 10 variables for ZDT1-ZDT3 functions, 5 variables for DTLZ2 function, 8 variables for DTLZ5 function, and 10 variables for DTLZ7. In contrast, these test cases are also optimized by the referenced MOEGO algorithm, which is run on a 12-core CPU (Intel Core i9-9900k) in parallel. The Monte Carlo sample points of the GMOEGO algorithm are set as  $n_p = 1 \times 10^5$  for two-objective functions (ZDT1-ZDT3), and  $n_p = 1 \times 10^6$  for three-objective functions (DTLZ2, DTLZ 5, and DTLZ7). Both algorithms are performed on the same computational platform mentioned at the beginning of this section.

The test results are shown in Table 2. The Pareto solution distributions compared with the theoretical true Pareto front are illustrated in Figure 7. It can be noticed that, for all the test cases, the Pareto solution distributions obtained by the GMOEGO algorithm and the referenced MOEGO algorithm are very close to the theoretical true Pareto front, and the obtained IGD values listed in the third column of Table 2 are around  $10^{-3}$  to  $10^{-5}$ , which indicates the optimized Pareto fronts are very close to the theoretical Pareto fronts.

**Table 2.** Results of the test functions.

Test Function	Algorithm	IGD	Total Cost (Hours)	MCHVI Cost (Hours)	SUR of Total Cost (Times)
ZDT1	MOEGO	$1.817 \times 10^{-5}$	4.952	4.511	5.246
	GMOEGO	$2.080 \times 10^{-5}$	0.944	0.703	
ZDT2	MOEGO	$2.124 \times 10^{-5}$	5.247	4.530	6.559
	GMOEGO	$3.878 \times 10^{-5}$	0.800	0.601	
ZDT3	MOEGO	$1.594 \times 10^{-4}$	4.249	2.474	4.634
	GMOEGO	$1.216 \times 10^{-4}$	0.917	0.636	
DTLZ2	MOEGO	$3.200 \times 10^{-3}$	32.943	32.843	13.734
	GMOEGO	$3.070 \times 10^{-3}$	2.397	2.365	
DTLZ5	MOEGO	$1.260 \times 10^{-4}$	28.014	26.138	11.906
	GMOEGO	$1.290 \times 10^{-4}$	2.353	2.272	
DTLZ7	MOEGO	$5.530 \times 10^{-3}$	29.679	27.426	12.408
	GMOEGO	$3.740 \times 10^{-3}$	2.392	2.183	

The total wall-clock time costs and the MCHVI calculation costs are given in the fourth and fifth columns of Table 2, and their corresponding comparison histories are illustrated in Figure 8. It can be seen that, as analyzed in Section 3, the computational cost of the MCHVI calculations is the main part of the overall optimization procedure, and it is not hard to imagine that improving this module of the optimization algorithm will bring the most benefits, and as expected, the computational cost improvements of the proposed GMOEGO algorithm mainly come from the speedup of the MCHVI calculations. The speedup ratio of the GMOEGO in comparison with the MOEGO algorithm is given in the sixth column of Table 2. It can be seen that benefit from the GPU implementation, speedups up to 13.734 times are achieved in comparison with the cost of the MOEGO algorithm based on 8 cores in parallel, which shows the effectiveness of the GPU architecture.

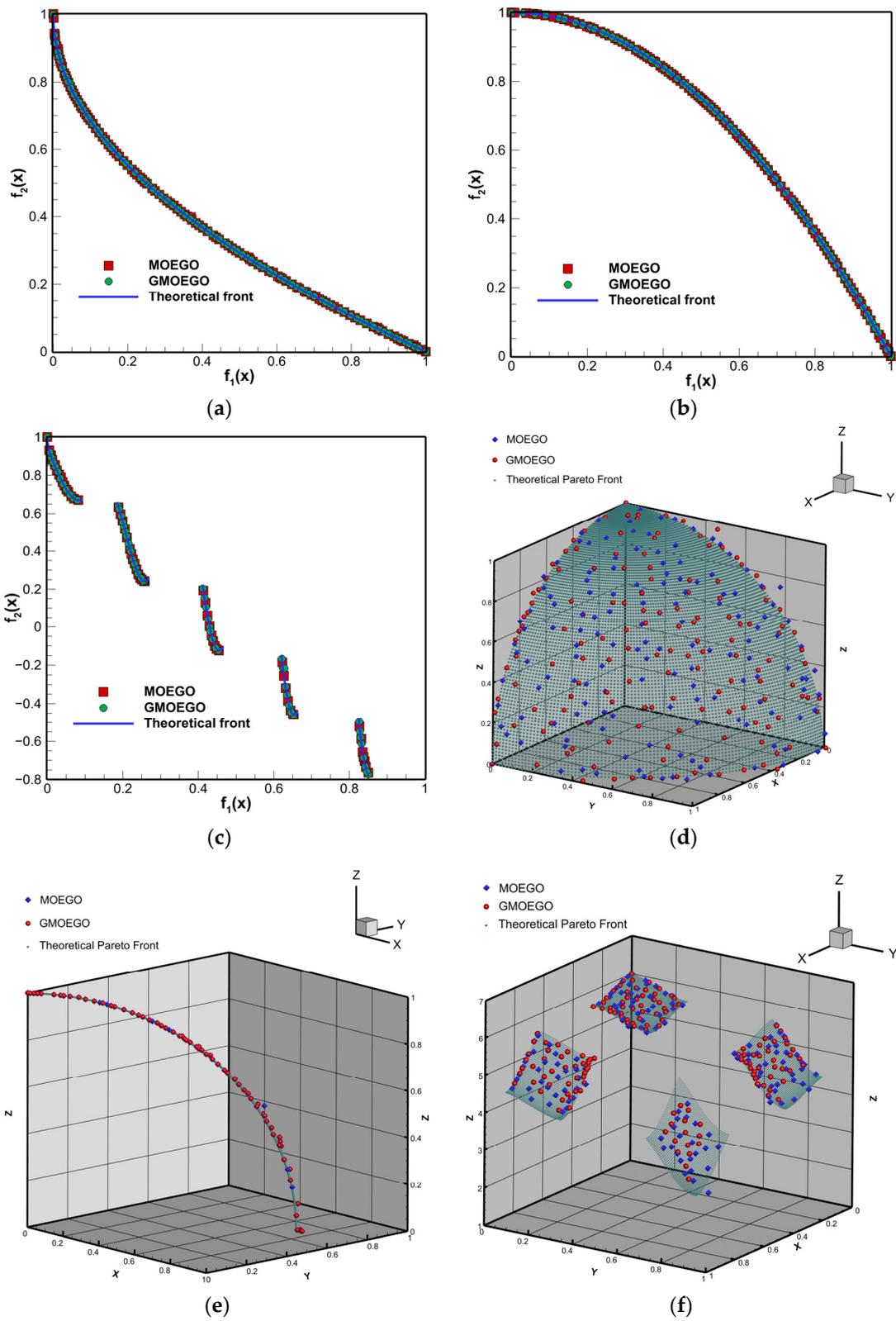


Figure 7. Pareto front comparisons: (a) ZTD1; (b) ZTD2; (c) ZTD3; (d) DTLZ2; (e) DTLZ5; (f) DTLZ7.

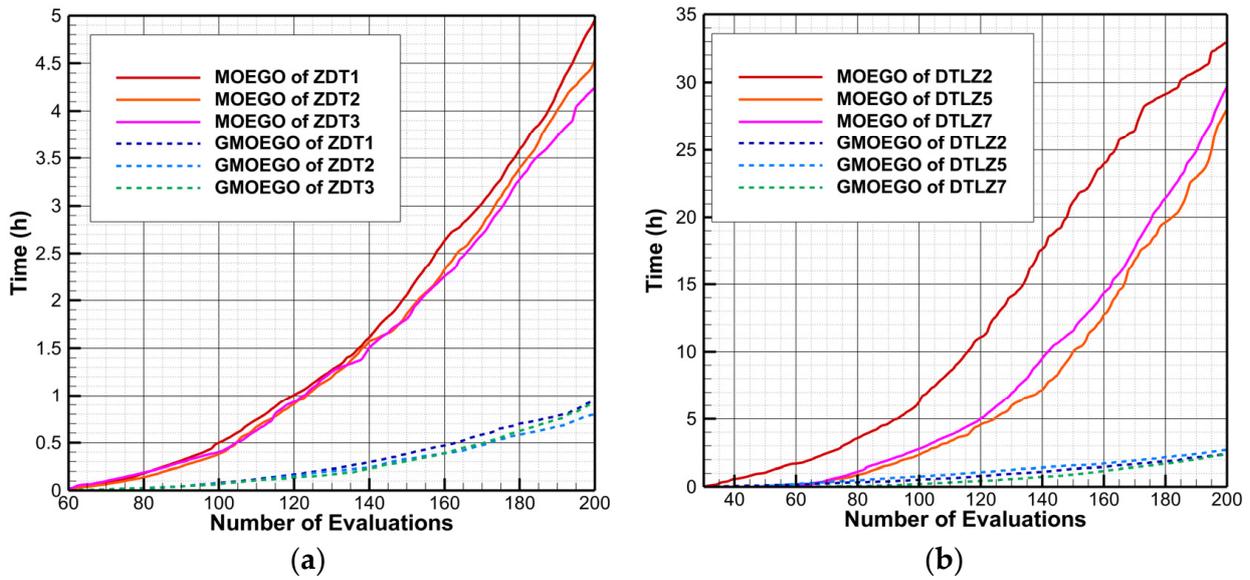


Figure 8. Wall-clock time history comparisons: (a) ZDT series functions; (b) DTLZ series functions.

### 5. Aerodynamic Design Optimization

In order to investigate the performance of the proposed GMOEGO algorithm in real engineering optimization applications, an aerodynamic airfoil shape optimization case [46] is selected to test the algorithm. The baseline of the airfoil is RAE2822 under the conditions of  $Ma = 0.75$ ,  $Re = 6.5 \times 10^6$ , and  $\alpha = 2.31^\circ$ . The optimization target is to maximize the lift-drag ratios under two different Mach numbers:  $Ma = 0.6$ ,  $Ma = 0.75$ . The optimization problem can be defined as:

$$\begin{aligned}
 &Max. \left. \begin{aligned} &(C_L/C_D)_{Ma=0.6} \\ &(C_L/C_D)_{Ma=0.75} \end{aligned} \right\} \\
 &s.t. \{x_i \in X | i = 1, 2, \dots, n\}
 \end{aligned} \tag{18}$$

in which  $C_L$  is the lift coefficient,  $C_D$  is the drag coefficient,  $X$  is the design parameters of the Hicks–Henne parametrization method [47], and  $n$  is the number of control parameters. In this paper,  $n = 14$  is set to parametrize the airfoil geometry.

The Computational Fluid Dynamics (CFD) simulator used to calculate the aerodynamic coefficients is the open-source software SU2 [48]. In this work, the Jameson scheme is selected as the spatial discretization scheme, and the Spalart–Allmaras (SA) one equation turbulence model is used for turbulence closure. The computational O-type mesh of RAE2822 airfoil is shown in Figure 9.

During the optimizations, 43 initial samples are set by using the LHS method, and the maximum FEs of CFD evaluations is limited to 400. The computations are carried out on the computational platform of Intel Core i9-9900k CPU (8 cores) with GTX-1066 GPU. After 400 CFD evaluations, the Pareto front obtained by the proposed GMOEGO algorithm is illustrated in Figure 10. Three representative optimized solutions marked with Optimum 1–3 in Figure 10 are selected to show their aerodynamic performances.

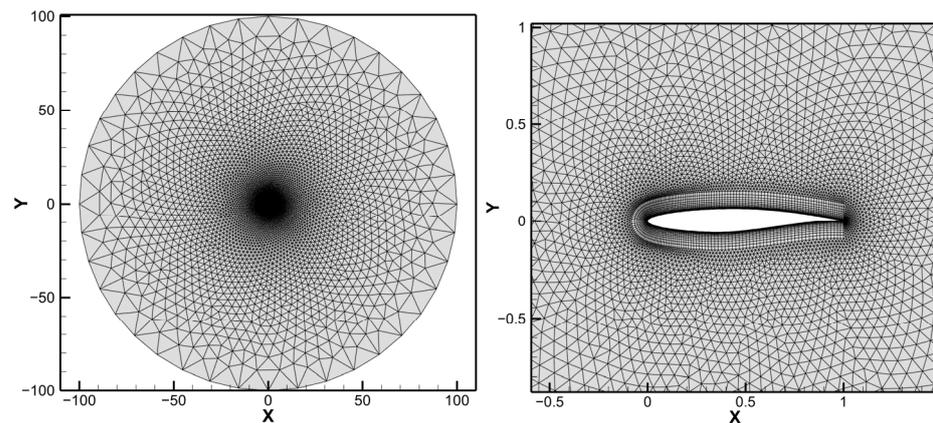


Figure 9. Computational mesh of RAE2822.

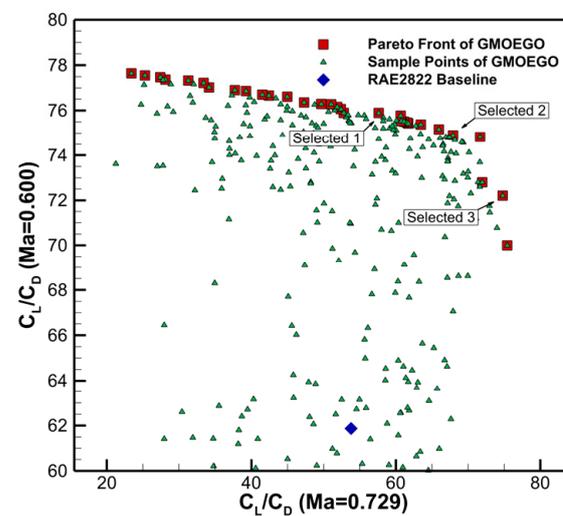


Figure 10. Optimization results of the GMOEGO algorithm.

The comparison of aerodynamic performance between the baseline and the selected optimized airfoils is listed in Table 3. It can be seen in the table that for optimum 1, the lift-drag ratio is increased by 7.14% for objective 1 and 22.57% for objective 2 when compared to the baseline. For optimum 2, the optimized ratios are 26.26% and 21.10%, respectively. For optimum 3, the lift-drag ratio of objective 1 is increased to 39.04%, and objective 2's improvement of lift-drag ratio is 16.73%.

The geometry shape of the optimized airfoil and the baseline are compared in Figure 11. Figure 12 illustrates the comparison of  $C_p$  distributions of two objectives, and the corresponding  $C_p$  comparisons are depicted in Figure 13. It can be learned that for objective 1, the shock wave of optimum 3 is weaker than that of optimums 1 and 2, which results in a significantly increasing lift-drag ratio (39.04%). For objective 2, the increase in lift-drag ratios is mainly due to the increase in lift coefficients. It can also be seen that different points on the Pareto front relate to the different aerodynamic performances of airfoils. Designers can select a specific airfoil depending on specific application scenarios and requirements.

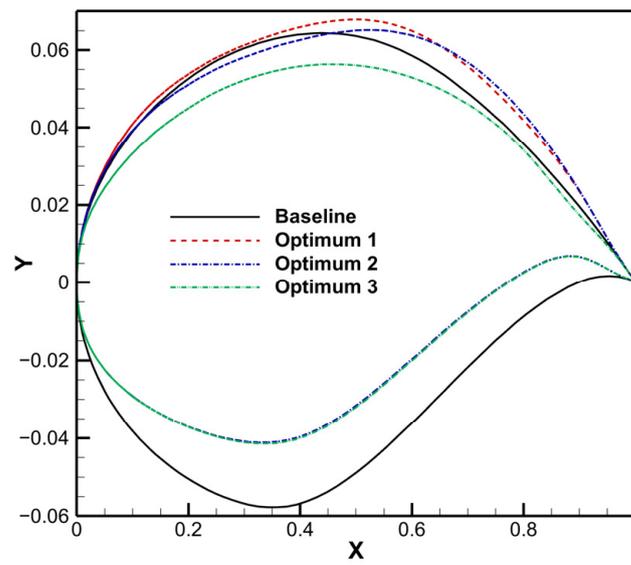


Figure 11. Comparison of shape between optimized airfoil and baseline.

Table 3. Comparison of aerodynamic performance between baseline and optimized airfoils.

Airfoil	Number of Objective	$C_L/C_D$
Baseline	1	53.81
	2	61.86
Optimum 1	1	57.65 (+7.14%)
	2	75.90 (+22.57%)
Optimum 2	1	67.94 (+26.26%)
	2	74.91 (+21.10%)
Optimum 3	1	74.82 (+39.04%)
	2	72.21 (+16.73%)

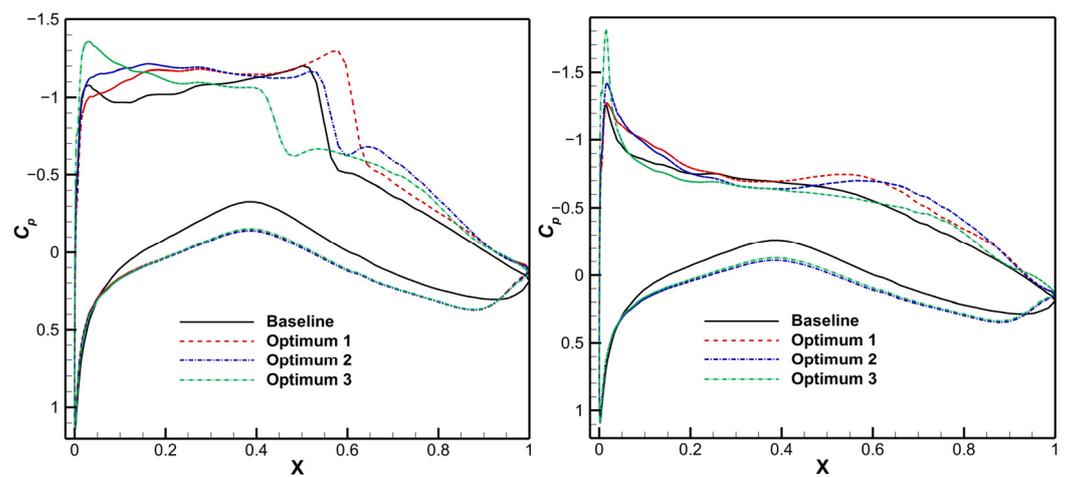
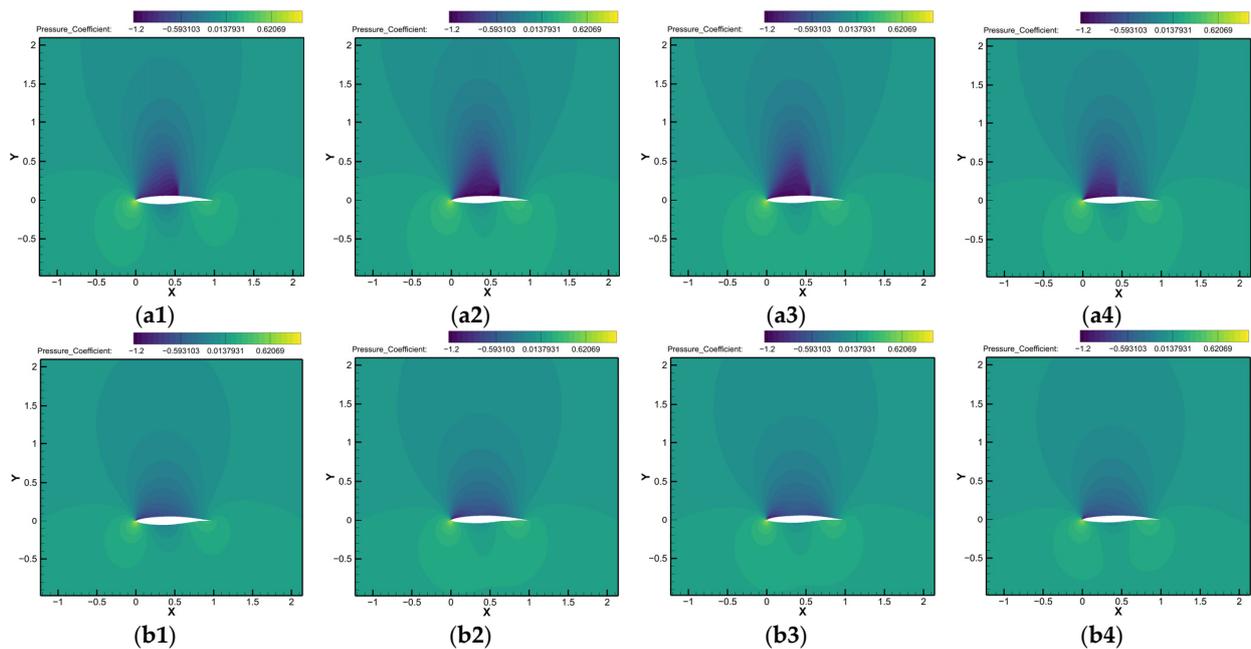


Figure 12. Comparison of pressure coefficient distribution between optimized airfoil and baseline.



**Figure 13.** Contour of pressure coefficient comparison between optimized airfoil and baseline: (a1) Baseline of objective 1; (a2) Optimum 1 of objective 1; (a3) Optimum 2 of objective 1; (a4) Optimum 3 of objective 1; (b1) Baseline of objective 2; (b2) Optimum 1 of objective 2; (b3) Optimum 2 of objective 2; (b4) Optimum 3 of objective 2.

Furthermore, in order to investigate the acceleration effect of the proposed GMOEGO algorithm in real applications, the aerodynamic airfoil shape optimization case above is also optimized by the counterpart referenced MOEGO algorithm. The computational platform is Intel Core i9-9900k CPU (8 cores), and all the other setups remain the same, as mentioned at the beginning of this section.

The optimized Pareto fronts obtained by MOEGO and GMOEGO algorithms for comparison is in Figure 14. It can be seen that both algorithms could obtain similar Pareto fronts. Figure 15 shows the comparison of the wall-clock time histories of the two algorithms, and the corresponding comparison results are listed in Table 4. The comparison shows that, benefiting from the Monte Carlo-based GPU-acceleration technique, a significantly speedup ratio, up to 7.27 times, is achieved. The fourth column of Table 4 shows the wall-clock time cost of MCHVI, and the corresponding histories are also depicted in Figure 15. It can be seen that the MCHVI calculation is the most time-consuming part of the whole optimization procedure of both algorithms, and the acceleration of the GMOEGO algorithm is mainly driven by the acceleration of this MCHVI calculation part. The significant speedup ratio shows the capability of the proposed GMOEGO algorithm to deal with real engineering optimizations.

**Table 4.** Speedup results of MOEGO and GMOEGO algorithms.

Algorithm	Platform	Total Cost (Hours)	MCHVI Cost (Hours)	SUR of Total Cost (Times)
MOEGO	Intel Core i9-9900k CPU (with 8 cores in parallel)	255.78	235.69	7.27
GMOEGO	GTX-1066	35.20	17.13	

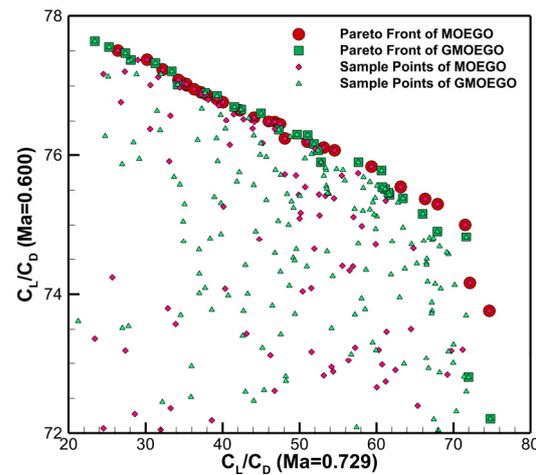


Figure 14. Obtained optimal Pareto fronts.

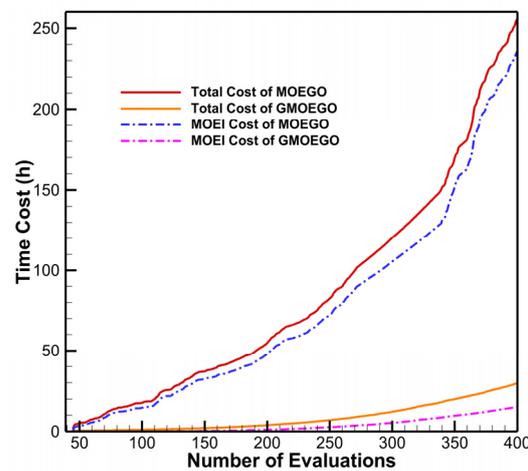


Figure 15. Time cost histories.

## 6. Conclusions

In this paper, the GMOEGO algorithm has been successfully developed for MOO problems. The outstanding hypervolume-based EI criterion is adopted as the multi-objective infill criterion computed in a novel Monte Carlo-based way of high efficiency. The time consumption of massive computing associated with Monte Carlo-based HVI approximation (MCHVI) is greatly alleviated by parallelizing in the GPU. The algorithm presented has been successfully validated by a set of typical mathematical test functions. Positive speedups up to 13.734 times can be achieved when compared with the traditional Monte Carlo-based MOEGO algorithm. The presented algorithm has also been successfully applied to an aerodynamic optimization problem, and a 7.27-times speedup is achieved with limited expensive-to-evaluate CFD evaluations, which shows the potential capability of the algorithm to deal with real engineering MOO problems in view of accuracy and efficiency.

**Author Contributions:** Conceptualization, S.X., J.Z. and H.C.; methodology, S.X.; software, S.X. and J.Z.; validation, S.X. and J.Z.; formal analysis, S.X.; investigation, S.X., Y.G. (Yunkun Gao) and Y.G. (Yisheng Gao); resources, S.X. and H.G.; data curation, J.Z. and X.J.; writing—original draft preparation, S.X.; writing—review and editing, S.X., H.C., Y.G. (Yunkun Gao) and Y.G. (Yisheng Gao); visualization, H.G. and X.J.; supervision, S.X.; project administration, S.X.; funding acquisition, S.X., J.Z., H.C. and Y.G. (Yunkun Gao). All authors have read and agreed to the published version of the manuscript.

**Funding:** This research was funded by the National Natural Science Foundation of China (grant number 12102185, 11972189 and 12102188), the China Postdoctoral Science Foundation (grant number 2021M701693), the Natural Science Foundation of Jiangsu Province (grant number BK20190391), and the Natural Science Foundation of Anhui Province (grant number 1908085QF260).

**Institutional Review Board Statement:** Not applicable.

**Informed Consent Statement:** Not applicable.

**Data Availability Statement:** Not applicable.

**Conflicts of Interest:** The authors declare no conflict of interest.

### Appendix A

Definitions of selected mathematical functions.

Name	Number of Objectives	Number of Variables	Function	Constraints
ZDT1	2	2, 10	$f_1 = x_1; f_2 = h \cdot g$ $g = 1 + 9 \sum_{i=2}^m \frac{x_i}{m-1}$ $h = 1 - \sqrt{f_1/g}$	$x_i \in [0, 1]$
ZDT2	2	10	$f_1 = x_1; f_2 = h \cdot g$ $g = 1 + 9 \sum_{i=2}^m \frac{x_i}{m-1}$ $h = 1 - (f_1/g)^2$	$x_i \in [0, 1]$
ZDT3	2	10	$f_1 = x_1; f_2 = h \cdot g$ $g = 1 + 9 \sum_{i=2}^m \frac{x_i}{m-1}$ $h = 1 - \sqrt{f_1/g} - (f_1/g) \sin(10\pi f_1)$	$x_i \in [0, 1]$
DTLZ2	3	5	$f_1 = (1 + g) \cos(x_1 \pi/2) \cos(x_2 \pi/2)$ $f_2 = (1 + g) \cos(x_1 \pi/2) \sin(x_2 \pi/2)$ $f_3 = (1 + g) \sin(x_1 \pi/2)$ $g = \sum_{i=3}^{10} (x_i - 0.5)^2$	$x_i \in [0, 1]$
DTLZ5	3	8	$f_1 = (1 + g) \cos(\theta_1 \pi/2) \cos(\theta_2 \pi/2)$ $f_2 = (1 + g) \cos(\theta_1 \pi/2) \sin(\theta_2 \pi/2)$ $f_3 = (1 + g) \sin(\theta_1 \pi/2)$ $\theta_1 = x_1; \theta_2 = \frac{\pi}{4(1+g)}(1 + 2g x_2)$ $g = (x_3 - 0.5)^2$	$x_i \in [0, 1]$
DTLZ7	3	10	$f_1 = x_1; f_2 = x_2$ $f_3 = (1 + g)h$ $h = 3 - \sum_{i=1}^2 \left[ \frac{f_i}{1+g} (1 + \sin(3\pi f_i)) \right]$ $g = 1 + 9x_3$	$x_i \in [0, 1]$

### References

- Jones, D.R.; Schonlau, M.; Welch, W.J. Efficient Global Optimization of Expensive Black-Box Functions. *J. Glob. Optim.* **1998**, *13*, 455–492. [\[CrossRef\]](#)
- Emmerich, M.T.M.; Giannakoglou, K.C.; Naujoks, B. Single- and multiobjective evolutionary optimization assisted by Gaussian random field metamodells. *IEEE Trans. Evol. Comput.* **2006**, *10*, 421–439. [\[CrossRef\]](#)
- Forrester, A.I.J.; Keane, A.J. Recent advances in surrogate-based optimization. *Prog. Aerosp. Sci.* **2009**, *45*, 50–79. [\[CrossRef\]](#)
- Deng, F.; Qin, N.; Liu, X.; Yu, X.; Zhao, N. Shock control bump optimization for a low sweep supercritical wing. *Sci. China Technol. Sci.* **2013**, *56*, 2385–2390. [\[CrossRef\]](#)
- Xu, S.; Chen, H. Nash game based efficient global optimization for large-scale design problems. *J. Glob. Optim.* **2018**, *71*, 361–381. [\[CrossRef\]](#)
- Whitley, D. A genetic algorithm tutorial. *Stat. Comput.* **1994**, *4*, 65–85. [\[CrossRef\]](#)
- Sóbester, A.; Leary, S.J.; Keane, A.J. On the Design of Optimization Strategies Based on Global Response Surface Approximation Models. *J. Glob. Optim.* **2005**, *33*, 31–59. [\[CrossRef\]](#)

8. Horowitz, B.; Guimarães, L.J.d.N.; Dantas, V.; Afonso, S.M.B. A concurrent efficient global optimization algorithm applied to polymer injection strategies. *J. Pet. Sci. Eng.* **2010**, *71*, 195–204. [[CrossRef](#)]
9. Han, Z.-H.; Görtz, S.; Zimmermann, R. Improving variable-fidelity surrogate modeling via gradient-enhanced Kriging and a generalized hybrid bridge function. *Aerosp. Sci. Technol.* **2013**, *25*, 177–189. [[CrossRef](#)]
10. Chung, I.-B.; Park, D.; Choi, D.-H. Surrogate-based global optimization using an adaptive switching infill sampling criterion for expensive black-box functions. *Struct. Multidiscip. Optim.* **2018**, *57*, 1443–1459. [[CrossRef](#)]
11. Xu, S.; Chen, H.; Zhang, J. A study of Nash-EGO algorithm for aerodynamic shape design optimizations. *Struct. Multidiscip. Optim.* **2019**, *59*, 1241–1254. [[CrossRef](#)]
12. Knowles, J. ParEGO A hybrid algorithm with on-line landscape approximation for expensive multiobjective optimization problems. *IEEE Trans. Evol. Comput.* **2006**, *10*, 50–66. [[CrossRef](#)]
13. Keane, A.J. Statistical Improvement Criteria for Use in Multiobjective Design Optimization. *AIAA J.* **2006**, *44*, 879–891. [[CrossRef](#)]
14. Liu, W.; Zhang, Q.; Tsang, E.; Liu, C.; Virginas, B. On the Performance of Metamodel Assisted MOEA/D. In *ISICA 2007: Advances in Computation and Intelligence*; International Symposium on Intelligence Computation and Applications; Kang, L., Liu, Y., Zeng, S., Eds.; Springer: Berlin/Heidelberg, Germany, 2007; Volume 4683, pp. 547–557.
15. Namura, N.; Shimoyama, K.; Obayashi, S. Expected Improvement of Penalty-Based Boundary Intersection for Expensive Multiobjective Optimization. *IEEE Trans. Evol. Comput.* **2017**, *21*, 898–913. [[CrossRef](#)]
16. Qingfu, Z.; Wudong, L.; Tsang, E.; Virginas, B. Expensive Multiobjective Optimization by MOEA/D With Gaussian Process Model. *IEEE Trans. Evol. Comput.* **2010**, *14*, 456–474. [[CrossRef](#)]
17. Zitzler, E.; Thiele, L. Multiobjective optimization using evolutionary algorithms—A comparative case study. In *Parallel Problem Solving from Nature—PPSN V*; Lecture Notes in Computer Science; Eiben, A.E., Bäck, T., Schoenauer, M., Schwefel, H.-P., Eds.; Springer: Berlin/Heidelberg, Germany, 1998; Volume 1498, pp. 292–301.
18. Emmerich, M.T.M. Single- and Multi-objective Evolutionary Design Optimization Assisted by Gaussian Random Field Metamodels. Ph.D. Thesis, University of Dortmund, Dortmund, Germany, 2005.
19. Emmerich, M.; Beume, N.; Naujoks, B. An EMO Algorithm Using the Hypervolume Measure as Selection Criterion. In *The Third International Conference on Evolutionary Multi-Criterion Optimization, Berlin, Heidelberg*; Evolutionary Multi-Criterion Optimization; Springer: Berlin/Heidelberg, Germany, 2005; pp. 62–76.
20. Ponweiser, W.; Wagner, T.; Biermann, D.; Vincze, M. Multiobjective Optimization on a Limited Budget of Evaluations Using Model-Assisted S-Metric Selection. In *Parallel Problem Solving from Nature—PPSN X*; Springer: Berlin/Heidelberg, Germany, 2008; pp. 784–794.
21. Fleischer, M. The Measure of Pareto Optima Applications to Multi-objective Metaheuristics. In *Evolutionary Multi-Criterion Optimization*; Springer: Berlin/Heidelberg, Germany, 2003; pp. 519–533.
22. Wagner, T.; Emmerich, M.; Deutz, A.; Ponweiser, W. On Expected-Improvement Criteria for Model-based Multi-objective Optimization. In *Parallel Problem Solving from Nature, PPSN XI*; Springer: Berlin/Heidelberg, Germany, 2010; pp. 718–727.
23. Yang, K.; Deutz, A.H.; Yang, Z.; Bäck, T.; Emmerich, M. Truncated expected hypervolume improvement: Exact computation and application. In *Congress on Evolutionary Computation*; Springer: Berlin/Heidelberg, Germany, 2016; pp. 4350–4357.
24. Li, Z.; Wang, X.; Ruan, S.; Li, Z.; Shen, C.; Zeng, Y. A modified hypervolume based expected improvement for multi-objective efficient global optimization method. *Struct. Multidiscip. Optim.* **2018**, *58*, 1961–1979. [[CrossRef](#)]
25. Yang, K.; Emmerich, M.; Deutz, A.; Bäck, T. Multi-Objective Bayesian Global Optimization using expected hypervolume improvement gradient. *Swarm Evol. Comput.* **2019**, *44*, 945–956. [[CrossRef](#)]
26. Zitzler, E. Evolutionary Algorithms for Multiobjective Optimization Methods and Applications. Ph.D. Thesis, Swiss Federal Institute of Technology, Zürich, Switzerland, 1999.
27. Beume, N.; Rudolph, G. Faster S-Metric Calculation by Considering Dominated Hypervolume as Klee’s Measure Problem. In Proceedings of the Second IASTED International Conference on Computational Intelligence, San Francisco, CA, USA, 20–22 November 2006; pp. 233–238.
28. While, L.; Hingston, P.; Barone, L.; Huband, S. A Faster Algorithm for Calculating Hypervolume. *IEEE Trans. Evol. Comput. Optim. Appl.* **2006**, *10*, 29–38. [[CrossRef](#)]
29. Bradstreet, L.; While, L.; Barone, L. A Fast Incremental Hypervolume Algorithm. *IEEE Trans. Evol. Comput.* **2008**, *12*, 714–723. [[CrossRef](#)]
30. Bader, J.; Zitzler, E. HypE An algorithm for fast hypervolume-based many-objective optimization. *Evol. Comput.* **2011**, *19*, 45–76. [[CrossRef](#)]
31. Yang, K.; Emmerich, M.; Deutz, A.; Bäck, T. Efficient Computation of Expected Hypervolume Improvement Using Box Decomposition Algorithms. *J. Glob. Optim.* **2019**, *75*, 3–34. [[CrossRef](#)]
32. Krige, D.G. A statistical approach to some basic mine valuation problems on the Witwatersrand. *J. South Afr. Inst. Min. Metall.* **1951**, *52*, 119–139.
33. Sekishiro, M.; Venter, G.; Balabanov, V. Combined Kriging and Gradient-Based Optimization Method. In Proceedings of the 11th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference, Portsmouth, VA, USA, 6–8 September 2006; p. AIAA 2006-7091.
34. Jiang, S.; Zhang, J.; Ong, Y.S.; Zhang, A.N.; Tan, P.S. A Simple and Fast Hypervolume Indicator-Based Multiobjective Evolutionary Algorithm. *IEEE Trans. Cybern.* **2015**, *45*, 2202–2213. [[CrossRef](#)] [[PubMed](#)]

35. Hammersley, J.M.; Handscomb, D.C.; Weiss, G. Monte Carlo Methods. *Phys. Today* **1965**, *18*, 55. [[CrossRef](#)]
36. Evans, M.; Swartz, T. *Approximating Integrals via Monte Carlo and Deterministic Methods*; OUP Oxford: Oxford, UK, 2000.
37. Luo, C.; Shimoyama, K.; Obayashi, S. Kriging Model Based Many-Objective Optimization with Efficient Calculation of Expected Hypervolume Improvement. In Proceedings of the 2014 IEEE Congress on Evolutionary Computation (CEC), Beijing, China, 6–11 July 2014.
38. Feng, D. Research on Efficient Global Optimization Algorithm and Its Application. Ph.D. Thesis, Nanjing University of Aeronautics and Astronautics, Nanjing, China, 2011.
39. Ma, Z.; Wang, H.; Pu, S. GPU computing of compressible flow problems by a meshless method with space-filling curves. *J. Comput. Phys.* **2014**, *263*, 113–135. [[CrossRef](#)]
40. Zhang, J.; Chen, H.; Cao, C. A graphics processing unit-accelerated meshless method for two-dimensional compressible flows. *Eng. Appl. Comput. Fluid Mech.* **2017**, *11*, 526–543. [[CrossRef](#)]
41. Zhang, J.-L.; Ma, Z.-H.; Chen, H.-Q.; Cao, C. A GPU-accelerated implicit meshless method for compressible flows. *J. Comput. Phys.* **2018**, *360*, 39–56. [[CrossRef](#)]
42. NVIDIA. CUDA C++ Programming Guide, Version 10. 2010. Available online: <https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html> (accessed on 8 September 2022).
43. Kenny, Q.Y.; Li, W.; Sudjianto, A. Algorithmic construction of optimal symmetric Latin hypercube designs. *J. Stat. Plan. Inference* **2000**, *90*, 145–159.
44. Deb, K.; Thiele, L.; Laumanns, M.; Zitzler, E. Scalable Test Problems for Evolutionary Multi-Objective Optimization. In *Evolutionary Multiobjective Optimization. Advanced Information and Knowledge Processing*; Abraham, A., Jain, L., Goldberg, R., Eds.; Springer: London, UK, 2005.
45. Zhang, Q.; Li, H. MOEA/D: A multiobjective evolutionary algorithm based on decomposition. *IEEE Trans. Evol. Comput.* **2007**, *11*, 712–731. [[CrossRef](#)]
46. Zhang, X.S.; Liu, J.; Luo, S.B. An improved multi-objective cuckoo search algorithm for airfoil aerodynamic shape optimization design. *Hangkong Xuebao/Acta Aeronaut. Et Astronaut. Sinica* **2018**, *40*, 5.
47. Hicks, R.M.; Henne, P.A. Wing design by numerical optimization. *J. Aircr.* **1978**, *15*, 407–412. [[CrossRef](#)]
48. Economon, T.D.; Palacios, F.; Copeland, S.R.; Lukaczyk, T.W.; Alonso, J.J. SU2: An open-source suite for multiphysics simulation and design. *AIAA J.* **2016**, *54*, 828–846. [[CrossRef](#)]

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