Techniques for Accelerating Multi-Objective Evolutionary Algorithms in PlatEMO

1st Ye Tian  
Institutes of Physical Science and Information Technology, Anhui University, Hefei, China  
field910921@gmail.com

2nd Ran Cheng  
Department of Computer Science and Engineering, Southern University of Science and Technology, Shenzhen, China  
ranchengcn@gmail.com

3rd Xingyi Zhang  
School of Computer Science and Technology, Anhui University, Hefei, China  
xyzhanghust@gmail.com

4th Yaochu Jin  
Department of Computer Science, University of Surrey, Guildford, United Kingdom  
yaochu.jin@surrey.ac.uk

Abstract—It has been widely recognized that evolutionary computation is one of the most effective techniques for solving complex optimization problems. As a group of metaheuristics inspired by nature, the superiority of evolutionary algorithms is mainly attributed to the evolution of multiple candidate solutions, which can strike a balance between exploration and exploitation. However, the effectiveness of evolutionary algorithms is generally at the expense of efficiency, which reduces the prevalence of evolutionary algorithms in solving real-world optimization problems. In 2017, we proposed the evolutionary multi-objective optimization platform PlatEMO to facilitate the use of multi-objective evolutionary algorithms (MOEAs), where some delicate techniques were developed to improve the computational efficiency of MOEAs. These techniques have not been introduced before, since users need not care about them when using existing MOEAs or developing new MOEAs. To deepen the understanding of the core mechanisms of PlatEMO, this paper gives a comprehensive introduction to these techniques, including new non-dominated sorting approaches, matrix calculation, and parallel computing. Several comparative experiments are conducted for a quantitative understanding of the efficiency improvement brought by these techniques.

I. INTRODUCTION

Evolutionary algorithms broadly refer to a variety of metaheuristics inspired by the biological evolution mechanisms and swarm behaviors in nature, such as genetic algorithms [1], particle swarm optimization [2], and differential evolution [3]. These algorithms have been widely adopted in many scientific and engineering areas [4], [5], since they are versatile for solving different types of optimization problems. The success of evolutionary algorithms is mainly owing to the evolution of multiple candidate solutions (i.e., a population), which enables the algorithm to find optimal or sub-optimal solutions without any a priori knowledge about the problem. On the other hand, the computational complexity is considerably increased since each solution should be updated and reevaluated at each generation; in particular, some evolutionary algorithms even update each solution more than once [6], [7].

Therefore, evolutionary algorithms are often criticized for the low efficiency [8], and it becomes even more serious for multi-objective evolutionary algorithms (MOEAs) [9]. On the one hand, MOEAs have to evaluate each solution by calculating multiple objective functions rather than a single one. On the other hand, the selection operator of single-objective evolutionary algorithms can be achieved by directly comparing the objective values of solutions, whereas complex selection strategies should be designed in MOEAs for striking a balance between convergence and diversity, such as the non-dominated sorting in NSGA-II [10], the truncation method in SPEA2 [11], and the hypervolume based selection in HypE [12].

To address this issue, some work has been dedicated to improving the efficiency as well as the practicality of MOEAs from various aspects. For Pareto dominance based MOEAs whose computational resource is mainly consumed by determining the Pareto dominance relations between solutions, a number of non-dominated sorting approaches have been proposed to improve the efficiency of the sorting procedure. The time complexity of the original non-dominated sorting procedure is \(O(nm^2)\) with \(n\) denoting the population size and \(m\) denoting the number of objectives [13], which is then reduced to \(O(nm^2)\) by fast non-dominated sort [10] and \(O(n \log m \cdot n)\) by Jensen’s Sort [9]. Later, the time complexity is further reduced by a series of non-dominated sorting approaches [14]. For decomposition based MOEAs that divide the original problem into a number of simple subproblems to be solved simultaneously, parallelization scheme has been...
adopted to speed up the optimization of subproblems, such as the MOEA deployed on multiple CPUs [15] and the MOEA deployed on GPU [16]. For indicator based MOEAs that must repeatedly calculate the performance indicator, some efforts have been made to accelerate the calculation of hypervolume, which is an effective and popular performance indicator but has a super-polynomial time complexity [17]. For instance, HypE [12] suggests a hypervolume estimation algorithm by using Monte Carlo simulation, FV-MOEA [18] accelerates the calculation of the hypervolume contribution of each solution by ignoring irrelevant solutions, and R2HCA-EMOA [19] approximates the hypervolume contribution of each solution by using an R2 indicator.

In 2017, an evolutionary multi-objective optimization platform was proposed by us, called PlatEMO\(^1\) [20], which contains more than 100 open-source MOEAs and more than 200 open-source benchmark problems. More importantly, it provides a powerful GUI enabling users to implement comparative experiments over multiple MOEAs and multiple benchmark problems by one-click operation, where the statistical results can be directly saved as an Excel table or LaTeX table. PlatEMO also uses some delicate techniques to improve the computational efficiency of MOEAs, including non-dominated sorting, matrix calculation, and parallel computing. These techniques were embedded in public functions, which can be invoked by users to develop new MOEAs without understanding the implementation details of them. In this paper, we would like to give a comprehensive introduction to these techniques, hoping to help users deepen the understanding of the core mechanisms of PlatEMO and to provide the users inspirations for developing new MOEAs. To illustrate the benefits of these techniques, we conduct several experiments to quantitatively show the efficiency improvement brought by these techniques.

The rest of this paper is organized as follows. Section II introduces three non-dominated sorting approaches used in PlatEMO, and verifies their efficiency by comparing them to other non-dominated sorting approaches. Section III introduces the methods of using matrix calculation in MOEAs, and compares the runtimes of MOEAs with and without matrix calculation. Section IV introduces the way to deploy MOEAs on multiple CPUs in PlatEMO. Section V draws the conclusions.

II. COMPUTATIONALLY EFFICIENT NON-DOMINATED SORTING TECHNIQUES

Since the solutions for multi-objective optimization problems (MOPs) have multiple objective values rather than a single one, the Pareto dominance relation is used to compare the quality of two solutions. For minimization MOPs, solution x is said to Pareto dominate solution y (denoted by \(x \prec y\)) if and only if

\[
\begin{align*}
\forall i & \in \{1, \ldots, m\} : f_i(x) \leq f_i(y) \\
\exists j & \in \{1, \ldots, m\} : f_j(x) < f_j(y)
\end{align*}
\]

where \(f_i(x)\) denotes the \(i\)-th objective value of \(x\) and \(m \geq 2\) denotes the total number of objectives. In particular, a solution is called non-dominated if it is not dominated by any other solutions in a population. Based on the Pareto dominance relation, the idea of non-dominated sorting was suggested as the selection strategy [21] and implemented in NSGA [13] for the first time. Specifically, non-dominated sorting performs the following three steps to divide a population \(P\) into several subsets:

- **Step 1**: \(i = 1\).
- **Step 2**: Find all the non-dominated solutions from \(P\) and move them to the non-dominated front \(F_i\).
- **Step 3**: \(i = i + 1\); return to Step 2 until \(P\) is empty.

Obviously, the solutions in front \(F_i\) are better than those in front \(F_j\) for \(i < j\), as each solution in \(F_j\) must be dominated by at least one solution in \(F_i\). Since the NSGA-II [10] was proposed in 2002, a large number of MOEAs have adopted non-dominated sorting as the basic selection strategy in environmental selection, which are not limited to Pareto dominance based MOEAs [22]–[24] but also include decomposition based MOEAs [25], [26] and indicator based MOEAs [12], [27].

However, non-dominated sorting is relatively time-consuming since the dominance relation between each pair of solutions should be determined by (1). Therefore, a series of non-dominated sorting approaches have been proposed to increase the efficiency of non-dominated sorting [14]. PlatEMO adopts three non-dominated sorting approaches to be used in different cases, including efficient non-dominated sort with sequential search (ENS-SS) [28], tree-based efficient non-dominated sort (T-ENS) [29], and efficient non-dominated level update (ENLU) [30]. In what follows, the three non-dominated sorting approaches are briefly reviewed and verified by comparative experiments.

A. Efficient Non-Dominated Sort (ENS)

In contrast to the conventional non-dominated sorting procedure that sorts the solutions front by front, ENS [28] determines the front number of each solution in sequence, while the sorting results of the two procedures are the same. To be specific, ENS-SS performs the following three steps to divide a population \(P\) into several subsets:

- **Step 1**: Sort \(P\) in an ascending order of the first objective.
- **Step 2**: \(i = 1\); pick up the first solution \(x\) from \(P\).
- **Step 3**: If \(x\) is dominated by any solution in \(F_i\), \(i = i + 1\) and repeats Step 3; otherwise move \(x\) to \(F_i\) and return to Step 2 until \(P\) is empty.

\(^1\)PlatEMO 2.4: https://github.com/BIMK/PlatEMO
The core idea of ENS is to presort the population according to the first objective in Step 1, so that any solution $x$ will not be dominated by the solutions after it in the ascending order, and the front number of $x$ can be determined without checking the solutions after it. Hence, each solution needs to be compared with some solutions before it rather than all the solutions, and a large number of comparisons can be saved by ENS-SS. In short, ENS-SS is much efficient than fast non-dominated sort as evidenced by the empirical comparisons in [28].

B. Tree-Based Efficient Non-Dominated Sort (T-ENS)

As reported in [28], [29], the efficiency of ENS-SS considerably deteriorates in high-dimensional objective space [14], and the comparison between each pair of non-dominated solutions cannot be saved by ENS-SS. While most non-dominated sorting approaches including ENS-SS store the solutions in each non-dominated front by an array, a novel tree structure is designed in T-ENS [29] to address the curse of dimensionality, which enables the sorting approach to save many comparisons between non-dominated solutions. That is, ENS-SS can save comparisons by deducing $x \prec z$ from $x \prec y$ and $y \prec z$ owing to the transitivity of Pareto dominance, while T-ENS can also deduce $x \nprec z$ from $x \npreceq y$ and $y \npreceq z$ in some cases. As a result, T-ENS is more efficient than ENS-SS when the number of objectives is high.

C. Efficient Non-Dominated Level Update (ENLU)

There exist some MOEAs based on the steady-state evolution model, which generate a single offspring and update the population each time [31]. Assuming that the population size is $n$ and the number of generations is $g$, the non-dominated sorting should be performed $g$ times in general MOEAs, while it should be performed $n \times g$ times in steady-state MOEAs. As a result, steady-state MOEAs are generally less efficient than general MOEAs with the same number of function evaluations. To address this issue, ENLU [30] is tailored for steady-state MOEAs, which performs non-dominated sorting by taking advantage of the historical sorting result. More specifically, when an offspring is added to or deleted from the population, ENLU updates the non-dominated fronts rather than sorts the new population from scratch.

D. Empirical Studies

To illustrate the superiority of the above three non-dominated sorting approaches, several experiments are conducted to compare them to other existing non-dominated sorting approaches. Firstly, Fig. 1 depicts the runtime of best order sort (BOS) [32], corner sort [33], deductive sort [34], ENS-SS, fast non-dominated sort [10], and T-ENS on random populations with 2–4 objectives and 100–500 solutions, averaged over 20 runs. It is obvious that ENS-SS outperforms the other approaches on all the populations. By contrast, as shown in Fig. 2, T-ENS becomes more efficient than ENS-SS when the number of solutions and the number of objectives increase. One the other hand, Fig. 3 plots the runtime of ENS-SS, fast non-dominated sort, T-ENS, and ENLU on random populations with 2–10 objectives and 100–200 solutions. Note that the population is increased by new solutions steadily.
Secondly, the efficiency of the three non-dominated sorting approaches is investigated by embedding them into representative MOEAs. Table I lists the runtime of NSGA-III [35] embedded with six non-dominated sorting approaches on DTLZ2 [36], where the number of function evaluations is set to 20000, the population size is set to 100 and 500, and the other parameter settings are the same to those in [35]. The experiment uses NSGA-III rather than NSGA-II since the former can better solve the MOPs with many objectives. To make the number of reference points consistent with the population size, the mixture uniform design [37] is adopted to generate reference points instead of the original sampling method of NSGA-III. As can be observed from Table I, ENS-SS is the most efficient approach in most cases, while T-ENS becomes more efficient than ENS-SS when the number of objectives is more than 2 and the population size is 500. Moreover, Table II presents the runtime of MSEA [38] embedded with four non-dominated sorting approaches on DTLZ7 [36], where MSEA is a recently proposed steady-state MOEA having excellent diversity performance. The number of function evaluations is set to 20000 and the population size is set to 100 and 200. It can be found from Table II that the MSEA with ENLU consumes less runtime than MSEA with the other non-dominated sorting approaches.

As a consequence, the above experiments indicate that ENS-SS has the best efficiency when the number of objectives or the population size is small, T-ENS has the best efficiency when the number of objectives and the population size are large, and ENLU has the best efficiency for the steady-state evolutionary process. In PlatEMO, a function NSSort.m is provided for the non-dominated sorting of 77 MOEAs, where the sorting approach to be used is automatically determined based on the above experimental results. More specifically, ENS-SS is used when the number of objectives is 2 or the population size is smaller than 500, and T-ENS is used otherwise. Besides, a function UpdateFront.m is provided for the non-dominated sorting of four steady-state MOEAs, where ENLU is used.

### III. Matrix Calculation for Efficient Genetic Operators and Function Evaluations

Since matrix calculation can be accelerated by some programming languages (e.g., MATLAB) and hardware devices (e.g., GPU), it is desirable to improve the efficiency of MOEAs by matrix calculation. For example, the decision variables of all the solutions in a population can be represented by a matrix, where each row denotes a solution and each column denotes a decision variable. Thus, the generation of offsprings can be accelerated by performing genetic operators on the matrix, and the function evaluation can be accelerated by calculating the objective values of the matrix.

PlatEMO fits well with matrix calculation since it is fully developed in MATLAB, which provides various methods to accelerate matrix calculation. In the rest of this section, we introduce the three methods for accelerating matrix calculation in PlatEMO, including matrix operators, matrix functions, and GPU acceleration.

#### A. Matrix Operators

All the operators in PlatEMO are in fact conducted based on matrix calculation, including genetic operators (for real encoding [39], [40], binary coding [41], and permutation based encoding [42], [43]), particle swarm optimization [44] and its variants [45], [46], differential evolution [47], covariance matrix adaptation evolution strategy [48], and so on. In particular, the simulated binary crossover (SBX) [39] includes complicated formulas and procedures, which is taken as an example to illustrate the way to accelerate all its steps by matrix operators.

Given two parents $x_1 = (x_1^1, \ldots, x_1^d)$, $x_2 = (x_2^1, \ldots, x_2^d)$ and the crossover probability $p$, SBX performs the following steps to generate two offsprings $o_1 = (o_1^1, \ldots, o_1^d)$,

---

### TABLE I

<table>
<thead>
<tr>
<th>Number of objectives $m$, population size $n$</th>
<th>NSGA-III embedded with</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>BOS</td>
</tr>
<tr>
<td>$m = 2$, $n = 100$</td>
<td>1.3342</td>
</tr>
<tr>
<td>$m = 2$, $n = 500$</td>
<td>1.5586</td>
</tr>
<tr>
<td>$m = 3$, $n = 100$</td>
<td>1.5822</td>
</tr>
<tr>
<td>$m = 3$, $n = 500$</td>
<td>2.4367</td>
</tr>
<tr>
<td>$m = 5$, $n = 100$</td>
<td>1.9484</td>
</tr>
<tr>
<td>$m = 5$, $n = 500$</td>
<td>3.1479</td>
</tr>
<tr>
<td>$m = 10$, $n = 100$</td>
<td>1.9484</td>
</tr>
<tr>
<td>$m = 10$, $n = 500$</td>
<td>5.0384</td>
</tr>
</tbody>
</table>

### TABLE II

<table>
<thead>
<tr>
<th>Number of objectives $m$, population size $n$</th>
<th>MSEA embedded with</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ENS-SS</td>
</tr>
<tr>
<td>$m = 2$, $n = 100$</td>
<td>12.0792</td>
</tr>
<tr>
<td>$m = 2$, $n = 200$</td>
<td>15.4912</td>
</tr>
<tr>
<td>$m = 5$, $n = 100$</td>
<td>16.3255</td>
</tr>
<tr>
<td>$m = 5$, $n = 200$</td>
<td>20.7603</td>
</tr>
<tr>
<td>$m = 10$, $n = 100$</td>
<td>29.1788</td>
</tr>
<tr>
<td>$m = 10$, $n = 200$</td>
<td>32.4758</td>
</tr>
</tbody>
</table>
\( \mathbf{o}_2 = (o_2^1, \ldots, o_2^n) \), where \( \text{rand} \) denotes a random number within \([0, 1]\):

- **Step 1:** If \( \text{rand} > p \), let \( \mathbf{o}_1 = \mathbf{x}_1 \) and \( \mathbf{o}_2 = \mathbf{x}_2 \); otherwise go to Step 2.
- **Step 2:** For each dimension \( i \), if \( \text{rand} > 0.5 \), let \( o_1^i = x_1^i \) and \( o_2^i = x_2^i \); otherwise go to Step 3.
- **Step 3:** Calculate \( o_1^i \) and \( o_2^i \) by
  \[
  \begin{aligned}
  o_1^i &= [(1 + \beta)x_1^i + (1 - \beta)x_2^i] / 2, \\
  o_2^i &= [(1 - \beta)x_1^i + (1 + \beta)x_2^i] / 2,
  \end{aligned}
  \tag{2}
  \]

where

\[
\beta = \begin{cases} 
(2\mu)^{1/\eta}, & \mu \leq 0.5 \\
(2 - 2\mu)^{1/\eta}, & \mu > 0.5
\end{cases}
\tag{3}
\]

\( \mu \) is a random number within \([0, 1]\), and \( \eta \) is a parameter controlling the distribution of the offsprings.

- **Step 4:** If \( \text{rand} > 0.5 \), exchange \( o_1^i \) and \( o_2^i \); return to Step 2 until all the variables of the offsprings are generated.

Obviously, the procedure of SBX is quite complex that it contains two for-end blocks and three if-else blocks for generating \( 2n \) offsprings.

In order to improve the efficiency of SBX, PlatEMO uses four matrices \( \mathbf{X}_1, \mathbf{X}_2, \mathbf{O}_1, \) and \( \mathbf{O}_2 \) to represent the parents and offsprings, where each matrix has a size of \( n \times d \) with each row denoting a solution and each column denoting a decision variable. Generally, it is not difficult to calculate \( \mathbf{O}_1, \mathbf{O}_2 \) based on \( \mathbf{X}_1, \mathbf{X}_2 \) by (2) and (3), which can eliminate all the for-end blocks. But, it is not intuitive to eliminate the if-else blocks by matrix operators. Taking a closer look at (2), it can be found that the decision variables of the offsprings are mainly controlled by the value of \( \beta \). More specifically, the following assignments of \( \beta \) can lead to some special results:

- \( \beta = 1 \) will make \( o_1^i = x_1^i \) and \( o_2^i = x_2^i \);
- \( \beta = -1 \) will make \( o_1^i = o_2^i \) and \( o_2^i = o_1^i \).

Therefore, we can set \( \beta = 1 \) to prevent the crossover and set \( \beta = -1 \) to exchange the decision variables of the two offsprings. As a consequence, PlatEMO performs the following matrix calculations to achieve the same function to SBX, where all the involved matrix operators can be directly used in MATLAB:

- **Step 1:** Generate four random matrices \( \mathbf{M} \in [0, 1]^{n \times d} \), \( \mathbf{R}_1 \in [0, 1]^{n \times d} \), \( \mathbf{R}_2 \in [0, 1]^{n \times d} \), and \( \mathbf{R}_3 \in [0, 1]^{n \times 1} \).
- **Step 2:** Perform the following matrix calculations:
  \[
  \begin{align*}
  T_1 &= \mathbf{M} \leq 0.5, \quad 2 \\
  T_2 &= \text{sign}(\mathbf{R}_1 - 0.5), \quad 3 \\
  T_3 &= \mathbf{R}_2 > 0.5, \quad 6 \\
  T_4 &= \text{repmat}(\mathbf{R}_3 > p, d), \quad 4 \\
  B_1 &= (2 \times \mathbf{M})^{1/\eta}, \quad 5 \\
  B_2 &= (2 - 2 \times \mathbf{M})^{1/\eta}, \quad 9 \\
  B_3 &= \mathbf{B}_1 \cdot \mathbf{B}_1 + (1 - \mathbf{B}_1) \cdot \mathbf{B}_2, \quad 6 \\
  B_4 &= \mathbf{B}_2 \cdot \mathbf{B}_2, \quad 11 \\
  \mathbf{B}_5 &= (1 - \mathbf{B}_3) \cdot \mathbf{B} + \mathbf{T}_3, \quad 12 \\
  \mathbf{B}_6 &= (1 - \mathbf{B}_4) \cdot \mathbf{B} + \mathbf{B}_4, \quad 13 \\
  \mathbf{O}_1 &= [(1 + B) \cdot \mathbf{X}_1 + (1 - B) \cdot \mathbf{X}_2] / 2, \quad 14 \\
  \mathbf{O}_2 &= [(1 - B) \cdot \mathbf{X}_1 + (1 + B) \cdot \mathbf{X}_2] / 2. \quad 15
  \end{align*}
  \]

By doing so, the SBX only needs to perform several matrix calculations without any for-end block or if-else block, where (10) corresponds to (3), (11) corresponds to Step 4, (12) corresponds to Step 2, (13) corresponds to Step 1, and (14)(15) corresponds to (2). In PlatEMO, the above steps are implemented in the function GA.m.

### B. Matrix Functions

The above formulas adopt not only matrix operators but also some matrix functions provided by MATLAB (i.e., \( \text{sign()} \) and \( \text{repmat()} \)). When calculating the objective values of a population containing \( n \) solutions, PlatEMO also adopt several matrix functions to improve the efficiency. Taking the DTLZ2 [36] problem as an example, the objective values of a solution \( \mathbf{x}_1 = (x_1^1, \ldots, x_1^d) \) are calculated by

\[
\begin{align*}
  f_1(\mathbf{x}) &= (1 + g(\mathbf{x})) \cos \left( \frac{\pi x_1^1}{2} \right) \cdots \cos \left( \frac{\pi x_1^{d-2}}{2} \right) \cos \left( \frac{\pi x_1^{d-1}}{2} \right), \\
  f_2(\mathbf{x}) &= (1 + g(\mathbf{x})) \cos \left( \frac{\pi x_1^1}{2} \right) \cdots \sin \left( \frac{\pi x_1^{d-2}}{2} \right) \sin \left( \frac{\pi x_1^{d-1}}{2} \right), \\
  f_3(\mathbf{x}) &= (1 + g(\mathbf{x})) \cos \left( \frac{\pi x_1^1}{2} \right) \cdots \sin \left( \frac{\pi x_1^{d-2}}{2} \right), \\
  &\vdots \\
  f_n(\mathbf{x}) &= (1 + g(\mathbf{x})) \sin \left( \frac{\pi x_1^1}{2} \right)
\end{align*}
\tag{16}
\]

where

\[
g(\mathbf{x}) = \sum_{i=1}^{d} (x_i - 0.5)^2. \tag{17}
\]

Obviously, it is also not intuitive to replace the variables in the above formulas by matrices, since each objective function is related to different decision variables. Fortunately, the goal can be achieved by taking advantages of the matrix functions provided by MATLAB. More specifically:

- \( A < 0.5 \) returns a Boolean matrix, in which the value 1 denotes the corresponding element in \( A \) is smaller than 0.5 and 0 otherwise.
- \( \text{sign}(\mathbf{A}) \) returns a symbol matrix, in which the value 1 denote the corresponding element in \( A \) is larger than 0 and -1 otherwise.
- \( \text{repmat}(\mathbf{A}, d) \) returns a replicate matrix, in which \( A \) is repeated for \( d \) times along the column.
- The power operator works on each element rather than the whole matrix.
- The operator \( \cdot \) denotes the element-wise multiplication.
specifically, PlatEMO uses a matrix $X$ with size $n \times d$ to represent the decision variables of all the solutions in the population, and calculates the objective values on DTLZ2 by

$$G = \text{sum}((X^{m-d} - 0.5)^2), \quad ^7$$

$$F = \text{repmat}(1 + G, m) \cdot \text{flip}((\text{cumprod}((\text{cat}(I^n, \text{cos}((\pi/2) \cdot X^{1-m-1})))) \cdot (19))$$

The obtained matrix $F$ stores all the objective values of $X$, where each row denotes a solution and each column denotes an objective value. It can be seen that (18) corresponds to (17) and (19) corresponds to (16). In PlatEMO, the above steps are implemented in the function DTLZ2.m.

C. GPU Acceleration

The above matrix operators and matrix functions can also be accelerated by GPU, especially when the population size or the number of decision variables is large, e.g., when solving large-scale MOPs [49]. In MATLAB, all the matrices can be sent to GPU by the function \text{gpuArray()} and got back from GPU by the function \text{gather()}, while the matrix calculations on GPU uses the same matrix operators and matrix functions, i.e., the MATLAB codes do not need to be modified at all. However, it should be noted that the communication with GPU is also time-consuming, which can even decrease the efficiency if we send the matrices to GPU, perform matrix calculations on GPU, and get the matrices back from GPU. Instead, we should directly generate all the initial matrices (e.g., the decision variables of the initial population and the random matrices used in SBX) on GPU by the function \text{gpuArray.rand()} and make all the matrices stored in GPU all the time, thus saving the time consumed by the communication with GPU.

D. Empirical Studies

To verify the efficiency of matrix calculation based MOEA, Table III gives the runtime of NSGA-II on DTLZ2, where the number of function evaluations is set to 20000, the number of objectives is set to 3, the population size is set to 100, 500, 1000, and 5000. It can be seen from the table that the NSGA-II with matrix operators is more efficient than the original NSGA-II, while the efficiency can be further improved by matrix functions.

As for the NSGA-II based on GPU acceleration, it has the best efficiency when the number of decision variables is large. To summarize, the experiment indicates that the matrix operators and matrix functions can considerably improve the efficiency of MOEAs, and the efficiency can be further improved by GPU when solving large-scale MOPs.

For a comprehensive study, Fig. 4 compares the runtimes of the original NSGA-II and the NSGA-II accelerated by ENS-SS (i.e., non-dominated sorting), matrix operators (i.e., offspring generation), matrix operators (i.e., function evaluation), and GPU, where the number of function evaluations, the number of objectives, the population size, and the number of decision variables are set to 20000, 3, 500, and 5000, respectively. It is obvious that the runtime of the accelerated NSGA-II is approximately a quarter of the original version. As a consequence, the effectiveness of these techniques used...
in PlatEMO can be confirmed.

IV. PARALLELIZATION OF THE EXECUTION OF MOEAS

When comparing the performance of several MOEAs, it needs to execute each MOEA for multiple independent runs to eliminate the effect of random initialization and perform statistical analysis. To improve the efficiency of the experimental comparison between MOEAs, PlatEMO uses parallel computing to accelerate such a repetitive execution. More specifically, the parfor-end block is used to execute an MOEA for multiple runs instead of the for-end block, which can execute the loop in parallel by using all the CPUs in the computer.

To illustrate the efficiency improved by parallel computing, Fig. 5 depicts the runtime of executing NSGA-II and NSGA-III on DTLZ2 and DTLZ7 for 20 times. It can be found that the experiments performed in parallel are much more efficient than the experiments performed in sequence, where the parallel computing can reduce the runtime to a third. It is worth to note that the parallel computing is supported by four CPUs here, while the runtime cannot be reduced to a quarter since some runtime is consumed by the communication between CPUs.

In PlatEMO, the experimental module provided by the GUI can automatically perform the experiments in parallel. In other words, users can achieve parallel computing by one-click operation without writing any code.

V. CONCLUSIONS

Evolutionary algorithms have demonstrated high effectiveness in solving complex optimization problems, but they are often criticized for the low efficiency. As an evolutionary multi-objective optimization platform, PlatEMO provides more than 100 MOEAs and improves their efficiency by various techniques. This paper has given a detailed introduction to these techniques, including three non-dominated sorting approaches, matrix operators, matrix functions, GPU acceleration, and parallel computing. Moreover, several experiments have been conducted to quantitatively show the efficiency improved by these techniques.

The techniques provided by programming languages and hardware devices do improve the computational efficiency of MOEAs to some extent, while novel search strategies should be developed to fundamentally improve the efficiency of MOEAs in solving real-world MOPs, especially those with a large number of decision variables. Although some variable decomposition based MOEAs have been tailored for large-scale MOPs, they require a large number of function evaluations that highly increase the computational complexity. Therefore, some other strategies (e.g., Pareto optimal space learning) should be employed to solve large-scale MOPs with both high effectiveness and high efficiency.

REFERENCES
