Parallel Best Order Sort for Non-dominated Sorting: A Theoretical Study Considering the PRAM-CREW Model

Sumit Mishra[1,2] and Carlos A. Coello Coello[3]

[1]Department of Computer Science & Engineering,
Indian Institute of Information Technology Guwahati, Assam – 781015, INDIA
[2]Departamento de Computación, CINVESTAV-IPN, Mexico City, MEXICO
[3]Departamento de Sistemas, UAM-Azcapotzalco, Mexico City, MEXICO
Email: sumit@iiitg.ac.in, ccoello@cs.cinvestav.mx

***Abstract*—In the current paper we focus on parallelization of non-dominated sorting which is an essential step in Pareto-based multi-objective evolutionary algorithms. The parallel approaches can help to reduce the overall execution time of multi-objective evolutionary algorithms. Although there have been some proposals to parallelize non-dominated sorting algorithms, most of them have focused on the fast non-dominated sort algorithm proposed by Deb *et al.* This paper explores the scope of parallelism in a recently proposed approach known as Best Order Sort, which was proposed by Roy *et al.* We focus on two different ways of achieving parallelism in Best Order Sort. The time and space complexity of these two parallel schemes is also analyzed theoretically considering the PRAM CREW model.**

***Index Terms*—Multi-objective optimization, Dominance, Non-dominated sorting, Parallelism**

I. INTRODUCTION

Pareto-based multi- and many-objective evolutionary algorithms heavily rely on non-dominated sorting which is a process in which solutions are sorted and placed in different non-dominated fronts based on the dominance relationship between them. Let there be N solutions in population $\mathbb{P} = \{sol_1, sol_2, \dots, sol_N\}$ which need to be sorted into various non-dominated fronts. Let's consider that M objectives are associated with each solution. A solution $sol_i \in \mathbb{P}$ can be represented in M-dimensional objective space as $sol_i = \{f_1(sol_i), f_2(sol_i), \dots, f_M(sol_i)\}$. Here, $f_m(sol_i), 1 \leq m \leq M$ denotes the value of sol_i for the m^{th} objective. Without loss of generality, the minimization problem where all the objective values need to be minimized, is considered. In minimization problems, the dominance relation among solutions can be described as follows. A solution sol_i is said to dominate another solution sol_j denoted as $sol_i \prec sol_j$ *iff* the two following conditions are satisfied:

- $f_m(sol_i) \leq f_m(sol_j), \forall m \in \{1, 2, \dots, M\}$
- $f_m(sol_i) < f_m(sol_j), \exists m \in \{1, 2, \dots, M\}$.

When a solution sol_i does not dominate sol_j, this is represented as $sol_i \not\prec sol_j$. Two solutions sol_i and sol_j are said to be non-dominated when neither $sol_i \not\prec sol_j$ nor $sol_j \not\prec sol_i$. Non-dominated sorting can be formally defined as follows.

Definition 1 (Non-dominated Sorting). *Given N solutions $\{sol_1, sol_2, \dots, sol_N\}$, non-dominated sorting divides N solutions into $K(1 \leq K \leq N)$ fronts $\{F_1, F_2, \dots, F_K\}$ organized in decreasing order of their dominance such that*

- $\forall sol_i, sol_j \in F_k$: $sol_i \not\prec sol_j$ *and* $sol_j \not\prec sol_i$ $(1 \leq k \leq K)$
- $\forall sol \in F_k$, $\exists sol' \in F_{k-1}$: $sol' \prec sol$ $(1 < k \leq K)$

In these sorted fronts, F_1 is the front with highest dominance, F_2 is the front with the second highest dominance and so on.

Different non-dominated sorting algorithms [1]–[14] are available in the literature. Some of the most recent proposals focus on parallel versions. These parallel algorithms [12]–[14] have mainly focused on Fast non-dominated sort [1]. However, other proposals such as Jensen's approach [2], ENS [6], DCNS [8], [15] and BOS [7] among others, are also suitable for parallelization. In this paper, we focus on the parallelization of a rceent approach known as *Best Order Sort (BOS)* [7]. BOS is one of the most efficient approaches currently available which saves several dominance comparisons. We discuss the parallelism in BOS in two different ways and, consequently, two different parallel versions are proposed considering PRAM CREW model. The time complexity of these parallel versions is theoretically analyzed in different scenarios. The space complexity of these parallel versions is also analyzed.

The rest of the paper is organized as follows: Some of the approaches for non-dominated sorting are described in Section II. Best Order Sort along with its complexity is illustrated in Section III. In Section IV, we discuss the computing environment which is adopted for our parallel algorithm. Parallelism in BOS is explored in Section V. Another parallel approach is described in Section VI. Finally, Section VII concludes the paper and provides some future research paths.

II. PREVIOUS RELATED WORK

The proposed approaches for non-dominated sorting can be broadly classified into two categories – sequential and divide-and-conquer. First we discuss the sequential approaches. Srinivas *et al.* [16] proposed the so-called naive approach where a maximum of $N-1$ and a minimum of 1 comparisons between a pair of solutions can be performed. The worst and the best case time complexity of this approach is $\mathcal{O}(MN^3)$ and $\mathcal{O}(MN^2)$, respectively. The space complexity is $\mathcal{O}(N)$. Deb *et al.* [1] proposed fast non-dominated sort to improve the $\mathcal{O}(MN^3)$ time complexity. The time complexity of this approach is $\mathcal{O}(MN^2)$ with space complexity $\mathcal{O}(N^2)$. McClymont *et al.* [3] developed two approaches: Deductive sort and Climbing sort (the first one performs better than the second one). The dominance relationship is used to make this approach efficient. Deductive sort has worst case time complexity $\mathcal{O}(MN^2)$ and best case time complexity $\mathcal{O}(MN\sqrt{N})$.

Zhang *et al.* [6] proposed a framework called ENS (Efficient Non-dominated Sorting). Two approaches (ENS-SS and ENS-BS) were proposed based on the ENS framework. The worst case time complexity of both approaches is $\mathcal{O}(MN^2)$. The best case time complexity of ENS-SS is $\mathcal{O}(MN\sqrt{N})$ and that of ENS-BS is $\mathcal{O}(MN\log N)$. Bao *et al.* [11] developed a Hierarchical Non-dominated Sorting (HNDS) approach with worst and best case time complexity $\mathcal{O}(MN^2)$ and $\mathcal{O}(MN\sqrt{N})$, respectively. Some other approaches have noticed that for a solution to be inserted into a front, there is no need to compare this solution with all the solutions of that front. Best Order Sort (BOS) [7], T-ENS (Tree Based Efficient Non-dominated Sorting) [9] and ENS-NDT (Efficient Non-dominated Sorting Based on Non-Dominance Tree) [10] are such approaches. In T-ENS, a front is represented as a tree to reduce the number of dominance comparisons. The best case time complexity of T-ENS is $\mathcal{O}(^{MN\log N}/_{\log M})$ and the worst case time complexity is $\mathcal{O}(MN^2)$. ENS-NDT [10] is proposed by extending EBS-BS [6]. ENS-NDT has a best case time complexity $\mathcal{O}(MN\log N)$ and a worst case time complexity $\mathcal{O}(MN^2)$.

Mishra *et al.* [17] has modified BOS to handle duplicate solutions efficiently. Recently, the generalized version of BOS called Generalized Best Order Sort (GBOS) was proposed. This approach handles duplicate solutions efficiently and retains the comparison set concept of BOS [18]. Bounded Best Order Sort (BBOS) [19] is an improved version of BOS. The worst case time complexity of BBOS is $\mathcal{O}(MN^2)$ and the best case time complexity is $\mathcal{O}(MN\log N)$. Recently, Moreno *et al.* [20] improved the performance of non-dominated sorting by especially focusing on Best Order Sort. Their work is experimental. However, we have focused here on its theoretical aspects.

Now divide-and-conquer based approaches are discussed. A recursive approach was proposed by Jensen *et al.* [2] with time complexity $\mathcal{O}(N\log^{M-1}N)$. However, Jensen's approach is not always correct. Fortin *et al.* [4] modified Jensen's approach to make it correct. Fortin's approach has a worst case time complexity $\mathcal{O}(MN^2)$ and an average case complexity $\mathcal{O}(N\log^{M-1}N)$. Mishra *et al.* [8], [15] also proposed a framework called DCNS (Divide-and-conquer Based Non-dominated Sorting) with worst case time complexity $\mathcal{O}(MN^2)$ and best case time complexity $\mathcal{O}(MN\log N)$.

III. BEST ORDER SORT

We discuss first the serial version of Best order sort (BOS). We consider here the updated version of BOS[1] where duplicate solutions are also handled. BOS works in two phases. In the first phase, sorting of the solutions is performed based on every objective. In the second phase, rank is assigned to the solutions. Algorithm 1 summarizes the steps of BOS.

Algorithm 1 INITIALIZATION OF BOS

```
Input: $\mathbb{P}$: Set of $N$ solutions where each solution is associated with $M$ objectives
Output: Ranked solutions
// Global variables
$L^i_j \leftarrow \emptyset, \forall j = 1, 2, \ldots, M, \forall i = 1, 2, \ldots, N$   // Stores the solutions with rank $i$ whose rank has been assigned based on the $j^{th}$ objective
$isRanked_s \leftarrow$ FALSE, $\forall s \in \mathbb{P}$   // Solutions ranked or not
$NR \leftarrow 0$   // Stores the no. of solutions which have been ranked
$NF \leftarrow 1$   // Stores the no. of fronts which have been discovered
$R_s \leftarrow 0, \forall s \in \mathbb{P}$   // Store the rank of each solution
for each $j \in \{1, 2, \ldots, M\}$ do
    $Q_j \leftarrow$ Sort $N$ solutions based on the $j^{th}$ objective
```

Initially, $N \times M$ empty sets denoted by L^i_j are initialized, where $1 \leq j \leq M$ and $1 \leq i \leq N$. L^i_j stores the solutions with rank i which have been assigned based on the j^{th} objective. The ranking status of each solution s, whether it is ranked or not, is stored in a variable *isRanked*$_s$. Variable *NR* stores the number of solutions which have been ranked. Variable *NF* stores the number of discovered fronts. Variable R_s stores the rank of solutions s. Now, the solutions are sorted based on each of the M objectives in ascending order. The list Q_j stores the sorted order of the solutions based on the j^{th} objective. The i^{th} solution of list Q_j is denoted by $Q_j(i)$.

Algorithm 2 performs the actual non-dominated sorting. Here, M sorted lists are considered as a matrix of size $N \times M$ known as '*sorted matrix*' where the j^{th} column represents Q_j. For ranking purpose, the solutions are considered from this matrix in a row-wise manner, starting from the first column to the last one, until all the solutions are ranked. When all the solutions are ranked, the process of *sorted matrix* traversal stops. Let a solution s be traversed in the j^{th} column of the *sorted matrix*. Algorithm 2 first checks whether s has already been ranked or not (line 4). If s has been already ranked, then s is added to $L^{R_s}_j$ (line 5). Otherwise, s is ranked using Algorithm 3. Once s is ranked, *isRanked*$_s$ is set to 'TRUE' (line 8) so that s is not ranked again.

When a solution s is assigned the rank based on the j^{th} objective using Algorithm 3, then s is compared with $L^k_j (1 \leq k \leq NF)$ in a sequential manner. When s is non-dominated with respect to every solution of L^k_j, then s is assigned the rank k and s is added to L^k_j. If s is dominated by any of the solutions of $L^k_j (1 \leq k \leq NF)$, then the rank count is incremented by one and the updated rank count is the rank of solution s. After this, s is added to $L^{R_s}_j$.

The sorted order of the solutions based on each of the M objectives is stored in a list of size N. Thus, $\mathcal{O}(MN)$ space is required to store M sorted lists. $N \times M$ sets are also considered, which requires $\mathcal{O}(MN)$ space. Thus, the

---

[1] https://github.com/Proteek/Best-Order-Sort/

Algorithm 2 MAIN STEP OF BOS

```
Input: Sorted set of solutions, $Q_1, Q_2, \dots, Q_M$
Output: Rank of each solution $s \in \mathbb{P}$
for $i \leftarrow 1$ to $N$ do
    for $j \leftarrow 1$ to $M$ do
        $s \leftarrow Q_j(i)$                          // Consider $i^{th}$ solution from $Q_j$
        if $isRanked_s$ = TRUE then                     // s is ranked or not
            $L_j^{R_s} \leftarrow L_j^{R_s} \cup \{s\}$          // Include s to $L_j^{R_s}$
        else                                            // s is not ranked
            FINDRANK($s, j$)         // Find the rank of s based on the $j^{th}$ objective
            $isRanked_s \leftarrow$ TRUE
            $NR \leftarrow NR + 1$                      // Increase the no. of ranked solutions
    if $NR = N$ then                                    // All the solutions are ranked
        BREAK                                           // Non-dominated sorting finished
```

Algorithm 3 FINDRANK

```
Input: s: Solution which need to be ranked, j: List number where s has been found
Output: Rank of s
$ranked \leftarrow$ FALSE                               // Solution s is not yet ranked
for $k \leftarrow 1$ to $NF$ do                         // Check all the fronts sequentially
    $isDominates \leftarrow$ FALSE       // s is non-dominated with respect to $L_j^k$
    for $t \in L_j^k$ do                                // for all solutions in $L_j^k$
        $isDominates \leftarrow$ ISDOMINATES($t, s$)    // Check if t dominates s or not
        if $isDominates$ = TRUE then                    // t dominates s
            BREAK
    if $isDominates$ = FALSE then                       // s is non-dominated with $L_j^k$
        $R_s \leftarrow k$
        $ranked \leftarrow$ TRUE
        $L_j^{R_s} \leftarrow L_j^{R_s} \cup \{s\}$              // Include s in $L_j^{R_s}$
        BREAK
if $ranked$ = FALSE then                                // Solution s is not yet ranked
    $NF \leftarrow NF + 1$                              // Increase the no. of fronts
    $R_s \leftarrow NF$
    $L_j^{R_s} \leftarrow L_j^{R_s} \cup \{s\}$                  // Include s in $L_j^{R_s}$
```

space required by BOS is $\mathcal{O}(MN)$. The time complexity of BOS, $T(N,M)$ depends on the two steps: (i) sorting of the solutions based on each objective, $T_{\text{presort}}(N,M)$ and (ii) rank assignment, $T_{\text{rank}}(N,M)$.

$$T(N,M) = T_{\text{presort}}(N,M) + T_{\text{rank}}(N,M) \qquad (1)$$

The time complexity to sort the solutions based on every objective, *i.e.*, $T_{\text{presort}}(N,M)$ is $\mathcal{O}(MN\log N) + (M-1)\mathcal{O}(N\log N) = \mathcal{O}(MN\log N)$ considering heap sort [7], [18]. Now, we analyze the time complexity of BOS in two different scenarios. In these scenarios, different situations are considered.

A. Number of fronts is 1

We discuss the time complexity of BOS in three different situations when the number of fronts is 1.

1. All the solutions are the same (in terms of objective values): In this case, $Q_1, Q_2, \dots, Q_M$ have the same order of the solutions. Thus, the solution in every column of an individual row of the *sorted matrix* is the same. Thus, a solution is explored for the first time in the matrix when it is found in the first column and the solution is ranked. Thus, a solution which is explored first time in the i^{th} row, is compared with the already ranked $i-1$ solutions which have been ranked based on the first objective. Thus, the time complexity of the second phase is given by Eq. (2). The time complexity of BOS in this situation is given by Eq. (3).

$$T_{\text{rank}}(N,M) = \sum_{i=1}^{N} M(i-1) = M\frac{1}{2}N(N-1) = \mathcal{O}(MN^2) \qquad (2)$$

$$T(N,M) = \mathcal{O}(MN\log N) + \mathcal{O}(MN^2) = \mathcal{O}(MN^2) \qquad (3)$$

2. Worst Case: Let there be no duplicate solutions. In the worst case scenario, the first to the $(M-2)^{th}$ objective values of every solution is the same, however, the values of the last two objectives should be able to declare all the solutions as non-dominated. Here, $Q_1, Q_2, \dots, Q_{M-1}$ have the same order of the solutions and in Q_M, it is just the opposite to Q_1. Thus, the solution in the first to the $(M-1)^{th}$ column of an individual row of the *sorted matrix* are the same and the solution in the last column is different. Hence, two solutions are ranked in each row – the first one from the first column and the second one from the last column. So, all the solutions are ranked when they are traversed in the initial $N/2$ rows of the *sorted matrix*. Thus, a solution which is traversed for the first time in the j^{th} column ($j \in \{1, M\}$) of the i^{th} row, is compared with the already ranked $i-1$ solutions which have been ranked based on the j^{th} objective. Hence, the time complexity of the second phase is given by Eq. (4). The time complexity of BOS in this situation is given by Eq. (5).

$$\begin{aligned} T_{\text{rank}}(N,M) &= \sum_{i=1}^{N/2} M\,[2(i-1)] \\ &= M{}^1\!/_4 N(N-2) = \mathcal{O}(MN^2) \end{aligned} \qquad (4)$$

$$T(N,M) = \mathcal{O}(MN\log N) + \mathcal{O}(MN^2) = \mathcal{O}(MN^2) \qquad (5)$$

3. Best Case: In this scenario, traversal of the *sorted matrix* follows a particular pattern. Here, before a solution is traversed for the second time in the matrix, all the solutions should be traversed at least one time. So, M solutions are ranked in each row. Thus, all the solutions are ranked in initial N/M rows of the *sorted matrix*. When $M > N$, then initial M solutions of first row are ranked. Thus, a solution which is explored for the first time in the i^{th} row and the j^{th} column, is compared with the already ranked $i-1$ solutions which have been ranked based on the j^{th} objective. Hence, the time complexity of the second phase is given by Eq. (6). The time complexity of the second phase is $\mathcal{O}(N)$ when $M \geq N$ as all the solutions will be ranked in the first row itself. Hence, the time complexity of BOS in this situation is given by Eq. (7) when $M \geq N$.

$$T_{\text{rank}}(N,M) = \sum_{i=1}^{N/M} M\,[M(i-1)] = \frac{1}{2}N(N-M) \qquad (6)$$

$$T(N,M) = \mathcal{O}(MN\log N) + \mathcal{O}(N) = \mathcal{O}(MN\log N) \qquad (7)$$

B. Number of fronts is N

Here, $Q_1, Q_2, \dots, Q_M$ have the same order of the solutions. Thus, the solution in every column of an individual row of the *sorted matrix* are the same. Thus, a solution is explored for the first time in the matrix when it is found in the first column and the solution is ranked. Thus, a solution which is explored for the first time in the i^{th} row, is compared with the already ranked $i-1$ solutions which have been ranked based on the first objective. Thus, the time complexity of the second phase is given by Eq. (8). The overall time complexity of BOS is given by Eq. (9).

$$T_{\text{rank}}(N,M)=\sum_{i=1}^{N}M(i-1)=M\frac{1}{2}N(N-1)=\mathcal{O}(MN^2) \quad (8)$$

$$T(N,M)=\mathcal{O}(MN\log N)+\mathcal{O}(MN^2)=\mathcal{O}(MN^2) \quad (9)$$

IV. COMPUTING ENVIRONMENT FOR PARALLELISM

In this paper, we assume the PRAM CREW (Parallel random-access machine with Concurrent Read, Exclusive Write) model, as considered in [12]. This model is the earliest as well as one of the best-known model of parallel computation. In this model, the same memory location can be read by multiple processors simultaneously. However, the same memory location cannot be written at the same time. Since simultaneous write operations are not allowed, we design the algorithm in such a way that no concurrent write operations occur.

V. SCOPE OF PARALLELISM

BOS has two phases. The first phase involves sorting of the solutions based on the second to the M^{th} objective in a parallel manner. The solutions are sorted based on the first objective before sorting the solutions based on other objectives as the sorted order based on the first objective is used to decide the ordering when two solutions share a common value for second to the M^{th} objective. The worst case time complexity of the parallel version of the first phase is $\mathcal{O}(MN\log N)+\mathcal{O}(N\log N)=\mathcal{O}(MN\log N)$. In the best case, the time complexity is $\mathcal{O}(N\log N)+\mathcal{O}(N\log N)=\mathcal{O}(N\log N)$. To further improve the time complexity of the parallel version, we can use the parallel sorting algorithm while sorting the solutions. There are two ways to do this: (i) use parallel merge sort while sorting and (ii) use parallel merge sort and a parallel merge procedure in parallel merge sort.

- ***Parallel merge sort without parallelism in the merge procedure:*** Parallel merge sort requires $\mathcal{O}(N)$ time to sort N numbers. The overall worst case time complexity to sort the solutions based on the first to the M^{th} objective is $\mathcal{O}(MN)+\mathcal{O}(N)=\mathcal{O}(MN)$ and the time complexity in the best case is $\mathcal{O}(N)+\mathcal{O}(N)=\mathcal{O}(N)$.
- ***Parallel merge sort with parallelism in the merge procedure:*** Parallel merge sort requires $\mathcal{O}(\log^3 N)$ time to sort N numbers where the merge procedure also has parallelism. The overall worst case time complexity to sort the solutions based on the first to the M^{th} objective is $\mathcal{O}(M\log^3 N)+\mathcal{O}(\log^3 N)=\mathcal{O}(M\log^3 N)$ and the time complexity in the best case is $\mathcal{O}(\log^3 N)+\mathcal{O}(\log^3 N)=\mathcal{O}(\log^3 N)$.

Algorithm 4 INITIALIZATION OF PARALLEL BOS

```
Input: $\mathbb{P}$: Set of $N$ solutions where each solution is associated with $M$ objectives
Output: Ranked solutions
    // Global variables
    $L_j^i \leftarrow \emptyset, \forall i = 1,2,\ldots,N, \forall j = 1,2,\ldots,M$   // Stores the solutions with rank $i$ whose rank has been assigned based on the $j^{th}$ objective
    $isRanked_s[1,2,\ldots,M] \leftarrow$ FALSE, $\forall s \in \mathbb{P}$   // Solution $s$ ranked or not based on each of the objectives
    $NF_j \leftarrow 1, \forall j = 1,2,\ldots,M$   // Number of fronts discovered so far for the $j^{th}$ objective
    $R_s[1,2,\ldots,M] \leftarrow 0, \forall s \in \mathbb{P}$   // Rank of solution $s$ based on each of the objectives
    /* START OF PARALLEL SECTION */
    for each $j \in \{1,2,\ldots,M\}$ do
        $Q_j \leftarrow$ Sort $N$ solutions based on the $j^{th}$ objective
    /* END OF PARALLEL SECTION */
```

The steps of the parallel version of BOS are summarized in Algorithm 4. In the second phase of BOS, the solutions can be ranked based on different objectives simultaneously. A solution can be ranked a maximum of M times, corresponding to each of the M objectives. However, the rank of a solution will be the same even if it will be ranked based on different objectives. To avoid concurrent write to assign the same rank to a solution based on different objectives (by different processors), we store the rank of a solution s in an array $R_s[\ ]$ of size M. $R_s[j]$ indicates the j^{th} entry in the array and stores the rank of a solution s based on the j^{th} objective. To check the status of a solution s whether it is ranked or not based on any of the M objectives, an array $isRanked_s[\ ]$ is used whose length is M. 'TRUE' at the j^{th} position in this array signifies that s has been ranked based on the j^{th} objective. To keep the number of discovered fronts when the solutions are ranked based on a particular objective (say the j^{th}), a variable NF_j is used.

Actual non-dominated sorting is performed in Algorithm 5. To rank the solutions in a parallel manner, the solutions in each column of a particular row of the *sorted matrix* are ranked simultaneously. A solution s is ranked based on the j^{th} objective using Algorithm 6. Once the rank is assigned to the solution based on the j^{th} objective, $isRanked_s[j]$ is set to 'TRUE' (line 5). After assigning rank to the solution of each column of a particular row of the *sorted matrix*, we find whether every solution has been ranked based on at least one objective or not (line 7). If all the solutions have been ranked, then the process stops; otherwise, we rank the solutions of the next row of the *sorted matrix*.

In Algorithm 6, solution s is compared with $L_j^k(1 \leq k \leq NF_j)$ in a sequential manner. Once s is found to be non-dominated with respect to all the solutions of L_j^k (line 8), then s is assigned the rank k and s is added to L_j^k. If s is dominated by any of the solutions of $L_j^k(1 \leq k \leq NF_j)$ (line 13), then rank count NF_j is incremented by one and then, the updated rank count is the rank of solution s. After this, s is added to $L_j^{R_s[s]}$.

Check whether all the solutions have been ranked or not: We can check whether all the solutions have been ranked or not in a parallel manner considering $isRanked_s[\ ]$ array for each solution s. As the length of a particular array is M so we process this array at $\log M$ different levels using an 'OR' operation. At the l^{th} level, there will be $M/2^l$ 'OR' operations. A solution is said to be ranked even if it has been ranked based on at least one objective. So, the 'OR' operation is considered,

Algorithm 5 MAIN LOOP OF PARALLEL BOS

```
Input: Sorted set of solutions, $Q_1, Q_2, \ldots, Q_M$
Output: Rank of each solution $s \in \mathbb{P}$
for $i \leftarrow 1$ to $N$ do
    /* START OF PARALLEL SECTION */
    for $j \leftarrow 1$ to $M$ do
        $s \leftarrow Q_j(i)$                         // Take $i^{th}$ element from $Q_j$
        FINDRANK PARALLEL$(s, j)$    // Find the rank of $s$ based on the $j^{th}$ objective
        $isRanked_s[j] \leftarrow$ TRUE         // Rank has been assigned to $s$ based on the $j^{th}$ objective
    /* END OF PARALLEL SECTION */
    Find whether all the $N$ solutions in the population have been ranked or not using $isRanked_s[\ ]$ in a parallel manner
    if All the solutions have been ranked then
        BREAK
```

Algorithm 6 FINDRANK PARALLEL

```
Input: Solution $s$ and List number $j$
Output: Rank of $s$
$ranked \leftarrow$ FALSE               // Solution $s$ is not yet ranked based on the $j^{th}$ objective
for $k \leftarrow 1$ to $NF_j$ do           // for all discovered ranks based on the $j^{th}$ objective
    $isDominates \leftarrow$ FALSE                  // $s$ is non-dominated with $L_j^k$
    for $t \in L_j^k$ do                            // for all solutions in $L_j^k$
        $isDominates \leftarrow$ ISDOMINATES$(t, s)$     // Find whether $t$ dominates $s$ or not
        if $isDominates =$ TRUE then                 // $t$ dominates $s$
            BREAK
    if $isDominates =$ FALSE then                 // $s$ is non-dominated with $L_j^k$
        $R_s[j] \leftarrow k$               // Assign rank to $s$ based on the $j^{th}$ objective
        $ranked \leftarrow$ TRUE          // Rank is assigned to $s$ based on the $j^{th}$ objective
        $L_j^{R_s[j]} \leftarrow L_j^{R_s[j]} \cup \{s\}$               // Include $s$ to $L_j^{R_s[j]}$
        BREAK
if $ranked =$ FALSE then   // Solution $s$ is not yet ranked based on the $j^{th}$ objective
    $NF_j \leftarrow NF_j + 1$                 // Update rank count based on the $j^{th}$ objective
    $R_s[j] \leftarrow NF_j$                // Assign rank to $s$ based on the $j^{th}$ objective
    $L_j^{R_s[j]} \leftarrow L_j^{R_s[j]} \cup \{s\}$               // Include $s$ to $L_j^{R_s[j]}$
```

as its output is 'TRUE' when any of its inputs is 'TRUE'. After processing the array at the last level, we get a single value (either 'TRUE' or 'FALSE'). 'TRUE' indicates that the solution has been ranked and 'FALSE' indicates that the solution has not been ranked. The time required to process a particular array in a parallel manner is $\mathcal{O}(\log M)$ because various 'OR' operations at a level can be performed simultaneously. There are total of N such arrays, corresponding to N solutions. All these arrays can be processed simultaneously, so the time required to process all *isRanked*$_s[\ ]$ arrays in a parallel manner is $\mathcal{O}(\log M)$.

After processing each *isRanked*$_s[\ ]$ array we get single value. So, at the end we have N values corresponding to N such arrays. These N values are processed at $\log N$ levels using an 'AND' operation. At the l^{th} level, there will be $N/2^l$ 'AND' operations. After processing these N values, we get a single value (either 'TRUE' or 'FALSE') after the last level. 'TRUE' indicates that all the solutions have been ranked and 'FALSE' indicates that not all the solutions have been ranked. The time required to process these N values in a parallel manner is $\mathcal{O}(\log N)$ as all the 'AND' operations at a level can be performed simultaneously. Thus, the time to check whether all the solutions have been ranked or not using *isRanked*$_s[\ ]$ arrays for each solution s is $\mathcal{O}(\log M) + \mathcal{O}(\log N) = \mathcal{O}(\log N)$.

Example 1. *Consider a population* $\mathbb{P} = \{sol_1, sol_2, sol_3, sol_4\}$ *with* 4 *objectives. isRanked*$_s[\ ]$ *array for each solution* s *is shown in Fig. 1 where the length of each array is* 4 *as the number of objectives is* 4*. The value of each of these arrays are chosen randomly just to show the use of these arrays. Each of these arrays are processed in a parallel manner. Once these arrays are processed, we get four values corresponding to each array. These four values are then processed in a parallel manner at* $2 (= \log 4)$ *level to get a final value. As the final value is* 'FALSE' *not all the solutions are ranked.*

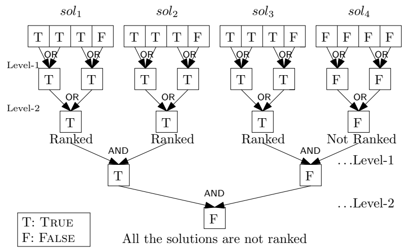

Fig. 1: Checking whether four solutions are ranked or not in a parallel manner. Four objectives are associated with each of the eight solutions

A. Parallelism in Dominance Comparisons

The dominance relationship between every pair of solutions can be obtained in advance in a parallel manner in $\mathcal{O}(M)$ time and stored in a $2D$ matrix (say M_{dom}) as done in [12]. We call this $2D$ matrix '*Dominance Matrix*'. The dominance comparison between a pair of solutions takes $\mathcal{O}(M)$ time and all the pairs of solutions can be compared simultaneously, so the time complexity to obtain the *dominance matrix* in a parallel manner is $\mathcal{O}(M)$. The $M_{\text{dom}}[i][j]$ cell of the dominance matrix stores the dominance relationship between sol_i and sol_j. The $\mathcal{O}(M)$ time complexity to obtain the dominance matrix can be further improved if a pair of solutions can be compared in less time. For this purpose, we create two Boolean arrays of size M. The first array (say A_{ij}) stores whether sol_i is better than sol_j for all the M objectives and the second array (say A_{ji}) stores whether sol_j is better than sol_i for all the M objectives.

In the process of checking whether sol_i is better than sol_j for all the M objectives, if the objective value of sol_i is better than (less than) the objective value of sol_j for the same objective, then the corresponding cell of the array A_{ij} is set to 'TRUE'; otherwise, it is set to 'FALSE'. In the same manner, array A_{ji} can also be filled. Both arrays A_{ij} and A_{ji} are processed in a parallel manner. As the size of the array is M, so these arrays are processed in $\log M$ levels. At each level, an 'OR' operation is performed between two consecutive array cells. At the l^{th} level, we perform $M/2^l$ 'OR' operations. After the 'OR' operation at the last level, we get either 'TRUE' or 'FALSE'.

After processing A_{ij} and A_{ji}, we have two values corresponding to these two arrays. Let the final value corresponding to A_{ij} and A_{ji} be denoted by *Result*$_1$ and *Result*$_2$ respectively. With the help of *Result*$_1$ and *Result*$_2$, the dominance relationship between sol_i and sol_j is identified as follows.

(i) **$Result_1 = Result_2 =$ FALSE:** Solutions sol_i and sol_j are the same (in terms of objective values).

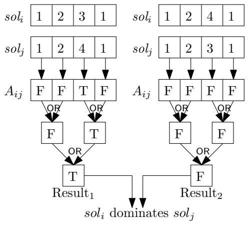

Fig. 2: Obtain the dominance relationship between sol_i and sol_j in a parallel manner

(ii) ***Result*$_1$ = *Result*$_2$ = TRUE:** Solutions sol_i and sol_j are non-dominated.
(iii) ***Result*$_1$ = TRUE and *Result*$_2$ = FALSE:** Solution sol_i dominates sol_j.
(iv) ***Result*$_1$ = FALSE and *Result*$_2$ = TRUE:** Solution sol_i is dominated by sol_j.

Arrays A_{ij} and A_{ji} can be filled in $\mathcal{O}(1)$ time in parallel. The time to process the arrays A_{ij} and A_{ji} in a parallel manner is $\mathcal{O}(\log M)$. The reason being that various 'OR' operations at an individual level can be performed in a parallel manner. Hence, it requires $\mathcal{O}(\log M)$ time to find the dominance relationship between every pair of solutions. Thus, the dominance matrix can be obtained in $\mathcal{O}(\log M)$ time in parallel. Now, we discuss the procedure to find the dominance relation between two solutions with the help of an example.

Example 2. *Let there be two solutions* $sol_i = \{1, 2, 3, 1\}$ *and* $sol_j = \{1, 2, 4, 1\}$ *in a* 4*-dimensional space. To find the dominance relationship between these two solutions, we compare* sol_i *with* sol_j *to fill array* A_{ij} *and compare* sol_j *with* sol_i *to fill array* A_{ji}*. These two arrays are shown in Fig. 2. These two arrays are processed in a parallel manner using 'OR'. After processing* A_{ij}*, we are getting* 'TRUE' *and after processing* A_{ji}*, we are getting* 'FALSE' *which means that* sol_i *dominates* sol_j*. The complete process to obtain the dominance relationship between the two solutions is shown in Fig. 2.*

Space Complexity to obtain the Dominance Matrix in Parallel: In this process, a solution is compared with another one using a Boolean array of size M. So, the space required is $\mathcal{O}(M)$. The number of pairs of the solutions is N^2 so, the space required to obtain the dominance matrix in parallel is $\mathcal{O}(MN^2)$. The size of the dominance matrix is $N \times N$ which requires $\mathcal{O}(N^2)$ space.

If the dominance matrix is obtained in advance in parallel in $\mathcal{O}(\log M)$ time, then it requires $\mathcal{O}(1)$ time to find the dominance relationship between the solutions as only a lookup in the dominance matrix will be required.

***Avoiding Checking ranked solutions**:* We can avoid checking whether all the solutions have been ranked or not after ranking the solutions of a particular row of the *sorted matrix*. For this purpose, we do not have to create an array of size M for each solution to store the rank, rather the rank of a solution can be stored in a single variable itself. In this case, when the solution is ranked, then while assigning rank to it, we have to check whether it is simultaneously ranked by other processors or not to avoid a write collision. This type of scenario is considered in [20]. However, in our work we are focusing on the algorithms where no such type of collision occurs. Now, we discuss the time complexity of the second phase of parallel BOS in two different scenarios in which the complexity is analyzed in the serial version of BOS.

B. Number of fronts is 1

We discuss the time complexity of the second phase of BOS in three different situations as discussed for the serial version of BOS.

1. All the solutions are the same: In this situation, the time complexity of rank assignment is given by Eq. (10).

$$\begin{aligned} T_{\infty\text{rank}}(N,M) &= \log M + \sum\nolimits_{i=1}^{N}(i-1) + \log N \\ &= \log M + {}^{1}\!/_{2}N(N-1) + N\log N \\ &= \mathcal{O}(\log M + N^2) \end{aligned} \tag{10}$$

2. Worst Case: Let there be no duplicate solutions. In this situation, the time complexity of rank assignment is given by Eq. (11).

$$\begin{aligned} T_{\infty\text{rank}}(N,M) &= \log M + \sum\nolimits_{i=1}^{N/2}(i-1) + \log N \\ &= \log M + {}^{1}\!/_{8}N(N-2) + {}^{1}\!/_{2}N\log N \\ &= \mathcal{O}(\log M + N^2) \end{aligned} \tag{11}$$

3. Best Case: In this situation, the time complexity of rank assignment is given by Eq. (12). This time complexity improves with an increase in the number of objectives. The time complexity is $\mathcal{O}(\log N)$ when $M \geq N$ as the first row solutions are traversed only.

$$\begin{aligned} T_{\infty\text{rank}}(N,M) &= \log M + \sum\nolimits_{i=1}^{N/M}(i-1) + \log N \\ &= \log M + \frac{1}{2M^2}N(N-M) + \frac{N}{M}\log N \end{aligned} \tag{12}$$

C. Number of fronts is N

The time complexity of rank assignment is given by Eq. (13).

$$\begin{aligned} T_{\infty\text{rank}}(N,M) &= \log M + \sum\nolimits_{i=1}^{N}(i-1) + \log N \\ &= \log M + {}^{1}\!/_{2}N(N-1) + N\log N \\ &= \mathcal{O}(\log M + N^2) \end{aligned} \tag{13}$$

D. Space Complexity

We discuss the space complexity of parallel BOS. In the first phase, merge sort is used which requires $\mathcal{O}(N)$ space to sort N solutions. As the solutions are sorted based on each of the M objectives individually, so the space required for the first phase is $\mathcal{O}(MN)$. The space required to store M sorted lists is $\mathcal{O}(MN)$. The space required to store $N \times M$ sets is $\mathcal{O}(MN)$. The space complexity to obtain the dominance matrix is $\mathcal{O}(MN^2)$ and the space complexity of storing the dominance matrix is $\mathcal{O}(N^2)$. As there are N solutions, so the space required to store the rank of all the solutions based on all the M objectives is $\mathcal{O}(MN)$. The space required to store *isRanked*$_s$[ ] arrays for all the N solutions is $\mathcal{O}(MN)$. The number of discovered fronts based on each of the M objectives requires $\mathcal{O}(M)$ space. Thus, the overall space required by the parallel version of BOS is $\mathcal{O}(MN^2)$.

VI. FURTHER IMPROVEMENT

In the previous parallel version, a solution s is compared with the solutions of L_j^k a in a sequential manner. However, this comparison can also be performed in a parallel manner.

Algorithm 7 PARALLEL FINDRANK-2

```
Input: Solution $s$ and List number $j$
Output: Rank of $s$
1:  ranked ← FALSE                // Rank is not yet assigned to $s$ based on the $j^{th}$ objective
2:  for $k \leftarrow 1$ to $NF_j$ do     // for all discovered ranks based on the $j^{th}$ objective
3:      isNondominated$[1, 2, \ldots, |L_j^k|]$ ← FALSE     // An array of length $|L_j^k|$ to store
        the dominance relation of $s$ with the solutions of $L_j^k$
        /* START OF PARALLEL SECTION */
4:      for $t \in L_j^k$ do                        // for all solutions in $L_j^k$
5:          isNondominated$[t]$ ← ISNON-DOMINATED$(t, s)$      // Find whether $t$ is
            non-dominated with $s$ or not
        /* END OF PARALLEL SECTION */
6:      Find whether $s$ is non-dominated with $L_j^k$   // This can be checked by processing
        the isNondominated[ ] array in a parallel manner
7:      if $s$ is non-dominated with every solution of $L_j^k$ then
8:          $R_s[j] \leftarrow k$                  // Assign rank to $s$ based on the $j^{th}$ objective
9:          ranked ← TRUE            // Rank is assigned to $s$ based on the $j^{th}$ objective
10:         $L_j^{R_s[j]} \leftarrow L_j^{R_s[j]} \cup \{s\}$                // Include $s$ to $L_j^{R_s[j]}$
11:         BREAK
12: if ranked = FALSE then  // Rank is not yet assigned to $s$ based on the $j^{th}$ objective
13:     $NF_j \leftarrow NF_j + 1$             // Update rank count based on the $j^{th}$ objective
14:     $R_s[j] \leftarrow NF_j$               // Assign rank to $s$ based on the $j^{th}$ objective
15:     $L_j^{R_s[j]} \leftarrow L_j^{R_s[j]} \cup \{s\}$                    // Include $s$ to $L_j^{R_s[j]}$
```

Algorithm 7 is used to assign rank to a solution s based on the j^{th} objective in an improved manner as compared to Algorithm 6. In this procedure, solution s is compared with $L_j^k (1 \leq k \leq NF_j)$ in a parallel manner. Solution s is compared with every solution of L_j^k simultaneously and the dominance relationship of s with every solution of L_j^k is stored in a Boolean array of size n_{jk} where $n_{jk} = |L_j^k|$. The comparison of s with every solution of L_j^k in a parallel manner can take $\mathcal{O}(1)$ time as we already have the *dominance matrix*. 'TRUE' in this Boolean array indicates that s is non-dominated with a particular solution of L_j^k, whereas 'FALSE' indicates that s is dominated by a particular solution of L_j^k. After obtaining the array, it is processed in a parallel manner at $\log n_{jk}$ levels using 'AND' operation between two consecutive values. The number of 'AND' operations at the l^{th} level is ${}^{n_{jk}}/_{2^l}$. After processing the array at the last level, we get a final value. 'TRUE' signifies that s is non-dominated with respect to every solution of L_j^k and 'FALSE' signifies that s is dominated by at least one of the solutions of L_j^k. All 'AND' operations can be performed at the same time at each level, so the time complexity to process the Boolean array is $\mathcal{O}(\log n_{jk})$.

When s is non-dominated with every solution of L_j^k (line 7), then s is assigned the rank k and s is added to L_j^k. If s is dominated by any of the solutions of $L_j^k (1 \leq k \leq NF_j)$ (line 12), then rank count NF_j is incremented by one and then, the updated value of rank count is the rank of solution s. After this, s is added to $L_j^{R_s[j]}$.

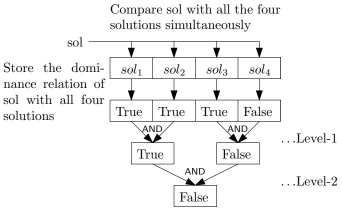
Compare sol with all the four solutions simultaneously
sol
Store the dominance relation of sol with all four solutions
sol_1
sol_2
sol_3
sol_4
True
True
True
False
AND
AND
True
False
...Level-1
AND
...Level-2
False

Fig. 3: Parallel comparison of *sol* with all the solutions of a front which are ranked based on the same objective.

Example 3. *Let there be four solutions* $\{sol_1, sol_2, sol_3, sol_4\}$ *which have been ranked (say based on the* j^{th} *objective) as shown in Fig. 3. Let's assume that there is a solution 'sol' which needs to be compared with these four solutions. In the previous parallel version, sol is compared with all these four solutions in a sequential manner. However, in this parallel version sol can be compared with all these four solutions simultaneously. After comparison, the dominance relation of sol with all the four solutions are stored in an array of size four. As the size of the array which stores the dominance relation is* 4*, so we process this array at* $\log 4$ *different levels using 'AND' operations. After processing the array at two levels, we are getting 'FALSE' which indicates that sol is dominated by at least one of these four solutions.*

We discuss the time complexity of rank assignment of the improved version of parallel BOS in two different scenarios.

A. Number of fronts is 1

The time complexity of rank assignment is discussed in three different situations as discussed for the serial version of BOS.

1. All the solutions are the same: When all the solutions are duplicate, then the time complexity of the rank assignment is obtained by Eq. (14).

$$\begin{aligned} T_{\infty\text{rank}}(N,M) &= \log M + 1 + \log N + \sum_{i=2}^{N} 1 + \lceil \log i \rceil + \log N \\ &= \log M + 2N \log N + 1 \\ &= \mathcal{O}(\log M + N \log N) \end{aligned} \tag{14}$$

2. Worst Case: Let there be no duplicate solutions, then the time complexity of the rank assignment in the worst case is given by Eq. (15).

$$\begin{aligned} T_{\infty\text{rank}}(N,M) &= \log M + 1 + \log N + \sum_{i=2}^{N/2} 1 + \lceil \log i \rceil + \log N \\ &= \log M + N \log N - {}^{N}/_{2} + 1 \\ &= \mathcal{O}(\log M + N \log N) \end{aligned} \tag{15}$$

3. Best Case: Time complexity of rank assignment in the best case is given by Eq. (16). When $M \geq N$, then the first row of the matrix is traversed and $T_{\infty\text{rank}}(N,M)$ becomes $\mathcal{O}(\log N)$.

$$\begin{aligned} T_{\infty\text{rank}}(N,M) &= \log M + 1 + \log N + \sum_{i=2}^{N/M} 1 + \lceil \log i \rceil + \log N \\ &= \log M + {}^{2N}/_{M} \log N - {}^{N}/_{M} \log M + 1 \end{aligned} \tag{16}$$

B. Number of fronts is N

The time complexity of rank assignment is given by Eq. (17).

$$\begin{aligned} T_{\infty\text{rank}}(N,M) &= \log M + \sum\nolimits_{i=1}^{N}(i-1) + \log N \\ &= \log M + {}^{1}\!/_{2}N(N-1) + N\log N \\ &= \mathcal{O}(\log M + N^2) \end{aligned} \tag{17}$$

C. Space Complexity

We discuss the space complexity of the improved parallel BOS. The improved version is the same as the previous parallel version except for the improved FINDRANK() procedure which is discussed in Algorithm 7. The space complexity of the previous parallel version is $\mathcal{O}(MN^2)$. In the improved FINDRANK() procedure, an array is considered in line 3 of Algorithm 7. The maximum size of this array can be N. As ranking is performed based on M objectives simultaneously, so this array can be used by each of the M processors. Hence, the space required to store this array is $\mathcal{O}(MN)$. Thus, the space complexity of the improved parallel version is $\mathcal{O}(MN^2)$.

Discussion: The time complexity of the parallel version of the first phase in the best case is $\mathcal{O}(\log^3 N)$ when all the solutions have a different value for the first objective. The best case time complexity of the parallel version of the second phase is $\mathcal{O}(\log N)$ (see Eqs. (12) and (16)). Thus, the best case time complexity of the parallel version of BOS is $\mathcal{O}(\log^3 N) + \mathcal{O}(\log N) = \mathcal{O}(\log^3 N)$. In [12], the time complexity of the parallel version of non-dominated sorting is proved to be $\mathcal{O}(M+N)$. The best case time complexity of the parallel version of BOS is $\mathcal{O}(\log^3 N)$ which is better than $\mathcal{O}(M+N)$ as proved in [12]. The worst case time complexity of the parallel BOS is $\mathcal{O}(\log M + N^2)$ (see Eqs. (13) and (17)) which is not good as compared to the time complexity reported in [12].

VII. CONCLUSIONS & FUTURE WORK

In the current paper, we have explored the scope of parallelism in an efficient algorithm for non-dominated sorting (BOS) in two different manners. We have analyzed the time complexity of two parallel versions in different scenarios. The space complexity is also analyzed. In this paper, we have focused on the parallelism from a theoretical point of view. In the future, we would like to find the scope of parallelism in other approaches as well. It would also be interesting to explore whether a divide-and-conquer based strategy can be utilized in BOS to explore its parallelization. An important question for non-dominated sorting is how to obtain the lower bound on its time complexity and we aim to pursue this goal.

ACKNOWLEDGEMENTS

The second author is on sabbatical leave from CINVESTAV-IPN. He gratefully acknowledges support from CONACyT grant no. 2016-01-1920 (*Investigación en Fronteras de la Ciencia 2016*).

REFERENCES

[1] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A Fast and Elitist Multiobjective Genetic Algorithm: NSGA–II," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, April 2002.

[2] M. T. Jensen, "Reducing the Run-Time Complexity of Multiobjective EAs: The NSGA-II and Other Algorithms," *IEEE Transactions on Evolutionary Computation*, vol. 7, no. 5, pp. 503–515, October 2003.

[3] K. McClymont and E. Keedwell, "Deductive Sort and Climbing Sort: New Methods for Non-dominated Sorting," *Evolutionary Computation*, vol. 20, no. 1, pp. 1–26, Spring 2012.

[4] F.-A. Fortin, S. Greiner, and M. Parizeau, "Generalizing the Improved Run-Time Complexity Algorithm for Non-dominated Sorting," in *2013 Genetic and Evolutionary Computation Conference (GECCO'2013)*. New York, USA: ACM Press, July 2013, pp. 615–622.

[5] H. Wang and X. Yao, "Corner Sort for Pareto-Based Many-Objective Optimization," *IEEE Transactions on Cybernetics*, vol. 44, no. 1, pp. 92–102, January 2014.

[6] X. Zhang, Y. Tian, R. Cheng, and J. Yaochu, "An Efficient Approach to Nondominated Sorting for Evolutionary Multiobjective Optimization," *IEEE Transactions on Evolutionary Computation*, vol. 19, no. 2, pp. 201–213, April 2015.

[7] P. C. Roy, M. M. Islam, and K. Deb, "Best Order Sort: A New Algorithm to Non-dominated Sorting for Evolutionary Multi-objective Optimization," in *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion*. Denver, Colorado, USA: ACM Press, July 20-24 2016, pp. 1113–1120.

[8] S. Mishra, S. Saha, and S. Mondal, "Divide and Conquer Based Non-dominated Sorting for Parallel Environment," in *2016 IEEE Congress on Evolutionary Computation (CEC'2016)*. Vancouver, Canada: IEEE Press, 24-29 July 2016, pp. 4297–4304.

[9] X. Zhang, Y. Tian, R. Cheng, and Y. Jin, "A Decision Variable Clustering-Based Evolutionary Algorithm for Large-Scale Many-Objective Optimization," *IEEE Transactions on Evolutionary Computation*, vol. 22, no. 1, pp. 97–112, February 2018.

[10] P. Gustavsson and A. Syberfeldt, "A New Algorithm Using the Non-dominated Tree to Improve Non-dominated Sorting," *Evolutionary Computation*, vol. 26, no. 1, pp. 89–116, September 2018.

[11] C. Bao, L. Xu, E. D. Goodman, and L. Cao, "A Novel Non-dominated Sorting Algorithm for Evolutionary Multi-Objective Optimization," *Journal of Computational Science*, vol. 23, pp. 31–43, November 2017.

[12] C. Smutnicki, J. Rudy, and D. Zelazny, "Very Fast Non-dominated Sorting," *Decision Making in Manufacturing and Services*, vol. 8, no. 1-2, pp. 13–23, 2014.

[13] S. Gupta and G. Tan, "A Scalable Parallel Implementation of Evolutionary Algorithms for Multi-Objective Optimization on GPUs," in *2015 IEEE Congress on Evolutionary Computation (CEC'2015)*. Sendai, Japan: IEEE Press, 25-28 May 2015, pp. 1567–1574.

[14] G. Ortega, E. Filatovas, E. M. Garzon, and L. G. Casado, "Non-dominated Sorting Procedure for Pareto Dominance Ranking on Multicore CPU and/or GPU," *Journal of Global Optimization*, vol. 69, no. 3, pp. 607–627, November 2017.

[15] S. Mishra, S. Saha, S. Mondal, and C. A. Coello Coello, "A Divide-And-Conquer Based Efficient Non-dominated Sorting Approach," *Swarm and Evolutionary Computation*, vol. 44, pp. 748–773, February 2019.

[16] N. Srinivas and K. Deb, "Multiobjective Optimization Using Nondominated Sorting in Genetic Algorithms," *Evolutionary Computation*, vol. 2, no. 3, pp. 221–248, Fall 1994.

[17] S. Mishra, S. Saha, and S. Mondal, "MBOS: Modified Best Order Sort Algorithm for Performing Non-dominated Sorting," in *2018 IEEE Congress on Evolutionary Computation (CEC'2018)*. Rio de Janeiro, Brazil: IEEE Press, July 8–13 2018, pp. 725–732.

[18] S. Mishra, S. Mondal, S. Saha, and C. A. Coello Coello, "GBOS: Generalized Best Order Sort Algorithm for Non-dominated Sorting," *Swarm and Evolutionary Computation*, vol. 43, pp. 244–264, December 2018.

[19] P. C. Roy, K. Deb, and M. M. Islam, "An Efficient Nondominated Sorting Algorithm for Large Number of Fronts," *IEEE Transactions on Cybernetics*, no. 99, pp. 1–11, 2018.

[20] J. Moreno, G. Ortega, E. Filatovas, J. Martínez, and E. Garzón, "Improving the Performance and Energy of Non-dominated Sorting for Evolutionary Multiobjective Optimization on GPU/CPU Platforms," *Journal of Global Optimization*, pp. 1–19, 2018.