Noname manuscript No.
(will be inserted by the editor)

Parallelism in Divide-and-Conquer Non-dominated Sorting: A Theoretical Study Considering the PRAM-CREW Model

Sumit Mishra · Carlos A. Coello Coello

Received: date / Accepted: date

Abstract Non-dominated sorting is a crucial component of Pareto-based multi- and many-objective evolutionary algorithms. As the number of objectives increases, the execution time of a multi-object-ive evolutionary algorithm increases, too. Since multi-objective evolutionary algorithms normally have a low data dependency, research-ers have increasingly adopted parallel programming techniques to reduce their execution time. Evidently, it is also desirable to parallelize non-dominated sorting. There are some recent proposals which focus on the parallelization of non-dominated sorting, with a particular emphasis on a very well-known approach called *fast non-dominated sorting*. In this paper, however, we explore the scope of parallelism in an approach called *Divide-and-Conquer based Non-dominated Sorting* (DCNS), which we recently introduced. This paper explores the parallelism from a theoretical point of view. The parallelization of the DCNS approach has been explored considering the PRAM-CREW (Parallel Random Access Machine, Concurrent Read Exclusive Write) model. The analysis of parallel algorithms is usually carried out under the assumption that an unbounded number of processors are available. So, in our analysis, we have also considered the same assumption. The time

Sumit Mishra
Departamento de Computación
CINVESTAV-IPN
Mexico City, D.F. 07360, Mexico
and
Department of Computer Science & Engineering
Indian Institute of Information Technology Guwahati
Guwahati – 781015, Assam, India
E-mail: smishra@computacion.cs.cinvestav.mx

Carlos A. Coello Coello
Departamento de Computación
CINVESTAV-IPN
Mexico City, D.F. 07360
Mexico
E-mail: ccoello@cs.cinvestav.mx

and space complexities of the parallel version of the DCNS approach is obtained in different scenarios. The time complexity of the parallel version of the DCNS approach in different scenarios is proved to be $\mathcal{O}(\log M + N)$. We have also obtained the maximum number of processors which can be required by the parallel version of the DCNS approach. The comparison of the parallel version of the DCNS approach with respect to some other approaches is also performed.

Keywords Non-dominated sorting · Dominance · Parallelism

1 Introduction

Non-dominated sorting is one of the most important components of Pareto-based multi- and many-objective evolutionary algorithms. Additionally, non-dominated sorting has applications in other domains such as economics, databases, game theory and computational geometry. In non-dominated sorting, the set of solutions known as population is divided into different non-dominated fronts. Let $\mathbb{P} = \{sol_1, sol_2, \ldots, sol_N\}$ be a set of N solutions which are in an M-dimensional space. In evolutionary algorithms, M is the number of objectives associated with each solution. We represent a solution $sol_i (1 \leq i \leq N)$ in M-dimensional objective space as $sol_i = \{f_1(sol_i), f_2(sol_i), \ldots, f_M(sol_i)\}$ where $f_m(sol_i), 1 \leq m \leq M$ is the value of sol_i for the m^{th} objective. In this paper, we assume (without loss of generalization) that all the objectives are to be minimized. In non-dominated sorting, the dominance relation between the solutions are considered, so we first discuss the dominance relation between the solutions.

Definition 1 (Dominance) *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:*

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

The notation $sol_i \not\prec sol_j$ is used to represent that solution sol_i does not dominate solution sol_j. Two solutions sol_i and sol_j are said to be non-dominated when neither dominates the other, *i.e.*, $sol_i \not\prec sol_j$ and $sol_j \not\prec sol_i$. Now, we formally define non-dominated sorting.

Definition 2 (Non-dominated sorting) *Non-dominated sorting divides the set of solutions $\{sol_1, sol_2, \ldots, sol_N\}$ in different non-dominated fronts $\{F_1, F_2, \ldots, F_K\}$ which are arranged in decreasing order of their dominance such that*

1. $\cup_{i=1}^{K} F_i = \mathbb{P}$
2. $\forall sol_i, sol_j \in F_k$*:* $sol_i \not\prec sol_j$ *and* $sol_j \not\prec sol_i$ $(1 \leq k \leq K)$
3. $\forall sol \in F_k$, $\exists sol' \in F_{k-1}$*:* $sol' \prec sol$ $(2 \leq k \leq K)$

In these sorted fronts, F_1 has the highest dominance, F_2 has the second highest dominance and so on. The last front F_K has the lowest dominance.

Nowadays, several researchers have focused on the parallel implementation of evolutionary algorithms [31], as different operations of the evolutionary algorithms are independent from each other. Many of the approaches have been recently proposed which focus on the parallelization of non-dominated sorting [8, 16, 21, 24, 28]. Some of these approaches [8, 24, 28] focus on *Fast Non-dominated Sorting* [4] for exploring the parallelism. Recently, Moreno *et al.* [21] proposed the parallelization of Best Order Sort (BOS) [26]. Two versions – a multicore based version and a GPU based version of BOS are presented. The parallel version of Efficient Non-dominated Sorting (ENS) [32] has been presented in [16]. The time complexity of the parallel version of ENS has also been obtained considering the PRAM-CREW model. However, there are other approaches which also have the parallelism property such as the naive approach [29], Jensen's Approach [11], Fang approach [6], DCNS [17], T-ENS [33] and ENS-NDT [9], among others.

Divide-and-conquer based algorithms are easy to parallelize [5] and there have been several approaches [2, 6, 7, 11, 17, 20] based on a divide-and-conquer strategy for non-dominated sorting. Some of these approaches [2, 7, 11] are based on the algorithm of Kung *et al.* [13]. Fortin *et al.* [7] has modified Jensen's algorithm [11] to remove its limitation and then a slightly modified version of Fortin *et al.* [7] is presented by Buzdalov *et al.* [2]. In Fortin's algorithm, there are two main procedures – NDHelperA and NDHelperB. Also, there are two splitting procedures – SplitA and SplitB. The NDHelperA procedure splits the problem into three sub-problems which need to be solved in a particular order. So, there is a sequential dependence in the NDHelperA procedure. The NDHelperB procedure also creates three sub-problems. However, in contrast to NDHelperA, these three sub-problems can be executed simultaneously. The approaches [2, 7, 11] are based on a divide-and-conquer strategy, however these have sequential dependency in the NDHelperA procedure.

Fang *et al.* [6] also developed a divide-and-conquer based algorithm. The dominance relationship among the solutions is hierarchical in nature [6], so a hierarchical data structure (a tree) is used in the approach. The dominance tree is adopted in this approach in which the population is recursively divided into two populations by the midpoint until the population contains only one solution. At this point, each of the single solutions can be considered as a dominance tree which has a single node. These dominance trees are merged to form a new dominance tree. The dominance tree of the entire population is formed when all the dominance trees are merged. From this final dominance tree, the non-dominated fronts of the population is obtained. In the approach, dividing the population is recursive in nature and also the process of merging the dominance trees is recursive. The dominance trees are merged in different rounds. Merging of different dominance trees in the same round can be performed simultaneously. So, parallelism is possible in this approach. However, the main concern is to have parallelism when merging two dominance trees.

DCNS [17] is an approach which is also based on a divide-and-conquer strategy. This approach works in two phases. In the first phase, the solutions

are sorted based on the first objective. The first phase can be implemented in a parallel manner considering a parallel sorting algorithm like parallel merge sort [3]. In the second phase, the solutions are assigned to different fronts considering a divide-and-conquer strategy. In the second phase, each solution is considered as a set of fronts which are merged at different levels. The second phase also has the parallelism property because different merge operations at the same level can be performed simultaneously. However, the parallelism of this kind in the DCNS approach is unimpressive. This is because of the serial merge procedure. However, the merge procedure is intrinsically prone to parallelization. In the DCNS approach, parallelism is possible in both phases. Also, the merge procedure itself has the parallelism property. So, we have lot of potential for parallelism in the DCNS approach. This motivated us to explore parallelism in the DCNS approach.

Parallelism has been explored considering the PRAM (Parallel Random Access Machine) model which is a natural extension of RAM (Random Access Machine). The RAM model is used for the analysis of sequential algorithms and PRAM is used for the analysis of parallel algorithms. In the case of the PRAM model, each processor is a Random Access Machine and these processors operate synchronously (*i.e.*, all processors follow a global clock). This model is the earliest as well as the best-known model of parallel computation [10, 12]. The analysis of parallel algorithms is usually carried out under the assumption that an unbounded number of processors is available [15, 22]. So, in our analysis, we have also considered the same assumption. In the analysis of the parallel approach, we obtain the maximum number of processors which can be required.

The main contributions of this paper are the following:

- The parallelism in the DCNS approach is explored in four different scenarios and those are theoretically analyzed in a thorough way. The time complexity of the serial and the parallel version of the DCNS approach is also analyzed in these scenarios. The maximum number of processors is also obtained.
- The best case time complexity of the serial version of the approach is proved to be $\mathcal{O}(N \log N + MN)$. In general, $M << \log N$, so we can say that the best case time complexity is $\mathcal{O}(N \log N)$.
- The time complexity of the parallel version is proved to be $\mathcal{O}(\log M + N)$.

The rest of this paper is organized as follows. The related work on non-dominated sorting is discussed in Section 2. The computing environment for parallelization is discussed in Section 3. The DCNS approach and the scope of parallelism in the approach are described in Section 4. The time complexities of the serial and parallel versions of DCNS in four different scenarios are obtained in Sections 5 – 8. The space complexity of the parallel version of DCNS is discussed in Section 9. The comparison of the parallel version of the DCNS approach with respect to the parallel version of some of other existing approaches is described in Section 10. Finally, Section 11 concludes the paper and provides some possible paths for future research.

2 Related Work

In this section, we discuss some of the approaches for non-dominated sorting that have been proposed so far. In the naive approach [29], a solution can be compared with respect to other solutions multiple times. The worst case time complexity of the naive approach is $\mathcal{O}(MN^3)$ and the best case time complexity is $\mathcal{O}(MN^2)$ with a space complexity $\mathcal{O}(N)$. In the naive approach, the dominance relation between different solutions can be computed simultaneously and stored in a matrix known as dominance matrix. A solution can also be compared with other solutions simultaneously. So, this approach also has the parallelism property. To reduce the worst case time complexity of the naive approach, fast non-dominated sorting [4] has been proposed with a time complexity $\mathcal{O}(MN^2)$ and a space complexity $\mathcal{O}(N^2)$. This approach also has the parallelism property which is discussed in [8, 24, 28]. A recursive approach is proposed by Jensen *et al.* [11]. The time complexity of this approach is $\mathcal{O}(N \log^{M-1} N)$ and the space complexity is $\mathcal{O}(MN)$. This approach is based on a divide-and-conquer strategy so it also has the parallelism property [5]. However, this approach is not applicable in the cases where two solutions share the same value for a particular objective. The best case time complexity of Jensen's approach is $\mathcal{O}(N \log N)$ which occurs when the number of objectives is two.

An approach based on a divide-and-conquer strategy was proposed by Fang *et al.* [6] with a worst case time complexity $\mathcal{O}(MN^2)$ and a best case time complexity $\mathcal{O}(MN \log N)$. The space complexity of this approach is $\mathcal{O}(MN)$. In case of duplicate solutions, one solution is considered as dominated by another. An approach based on arena's principle was proposed by Tang *et al.* [30] with a worst case time complexity $\mathcal{O}(MN^2)$. In some cases, the time complexity of this approach is $\mathcal{O}(MN\sqrt{N})$[32].

Climbing sort and deductive sort were proposed by McClymont *et al.* [14]. Deductive sort reduces the number of dominance comparisons by inferring the dominance relationship between the solutions. The worst case time complexity of deductive sort is $\mathcal{O}(MN^2)$ and the best case time complexity is $\mathcal{O}(MN\sqrt{N})$ with $\mathcal{O}(N)$ space complexity. The limitation of Jensen's approach was removed by Fortin *et al.* [7]. The worst case time complexity of this approach is $\mathcal{O}(MN^2)$. However, the average case time complexity remains $\mathcal{O}(N \log^{M-1} N)$.

Zhang *et al.* [32] developed an efficient non-dominated sorting (ENS) approach which consists of two phases. In the first phase, the solutions are sorted based on the first objective. In the second phase, the solutions are assigned to their respective fronts based on two different search techniques: sequential and binary. ENS-SS is based on sequential search and ENS-BS is based on binary search. The worst case time complexity of both ENS-SS and ENS-BS is $\mathcal{O}(MN^2)$. However, the best case time complexity of ENS-SS is $\mathcal{O}(MN\sqrt{N})$ and for ENS-BS is $\mathcal{O}(MN \log N)$. In this approach, the first phase has the parallelism property if parallel merge sort [3] or some other parallel sorting algorithms can be used to sort the solutions. In the second phase, a solution

can be compared with respect to all the solutions of a particular front simultaneously. However, when each front has a single solution, then this kind of parallelism cannot be achieved. The time complexity of non-dominated sorting was proved to be $\mathcal{O}(N \log^{M-1} N)$ by Buzdalov *et al.* [2]. Bao *et al.* [1] proposed a Hierarchical Non-dominated Sorting (HNDS). This approach also sorts the solutions based on the first objective, and then the solutions are assigned to their respective front as in ENS [32]. The best case time complexity of HNDS is $\mathcal{O}(MN\sqrt{N})$ and the worst case time complexity is $\mathcal{O}(MN^2)$ with space complexity $\mathcal{O}(N)$.

In many of the existing approaches, for a solution to be inserted into a front, it needs to be compared and non-dominated with respect to all the solutions in that front. Some of the approaches [9, 26, 33] that have been recently proposed do not have this requirement. Best order sort (BOS) [26] is one of such approaches which first sorts the solutions based on each objective. Then, the solutions are taken from this sorted list of solutions based on each objective to assign the rank. The worst case time complexity is $\mathcal{O}(MN^2)$ and the best case time complexity is $\mathcal{O}(MN \log N)$. This approach reduces the number of dominance comparisons to a great extent. This approach has been recently updated by the authors to handle duplicate solutions[1]. This approach has also been independently updated in [19]. There have been some approaches based on BOS. Generalized Best Order Sort (GBOS) [18] and Bounded Best Order Sort (BBOS) [27] are such approaches. In BOS, the sorting of solutions based on the second to the last objectives can be performed simultaneously. Also, when the solutions are sorted based on a particular objective, this process can be performed in parallel using parallel merge sort. In BOS, the solutions can be ranked based on different objectives simultaneously. While assigning a rank to a solution, a solution can be compared simultaneously with all the solutions that have been assigned the same rank based on a particular objective.

Zhang *et al.* [33] developed an approach known as T-ENS. Like ENS [32], here also, in the first phase, the solutions are sorted based on the first objective. In the second phase, the solutions are assigned to their respective fronts. A front is represented as a tree to avoid many comparisons. The worst case time complexity of T-ENS is $\mathcal{O}(MN^2)$ and the best case time complexity is $\mathcal{O}(MN \log N / \log M)$. However, this approach is not suitable for the cases where the solutions share identical values for any of the objectives [9]. By extending ENS-BS [32], an approach known as ENS-NDT [9] was developed which works in three phases. Here, in the first phase the solutions are sorted based on the last objective. In the second phase, prebalanced splits are created and in the third phase, solutions are assigned to their corresponding front. In this approach, duplicate solutions are handled efficiently. The best case time complexity of ENS-NDT is $\mathcal{O}(MN \log N)$ when $M > \log N$; otherwise, it is $\mathcal{O}(N \log^2 N)$. The worst case time complexity is $\mathcal{O}(MN^2)$. Few other approaches like [25, 27, 34] have been recently proposed for non-dominated sorting.

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

3 Computing Environment

This paper considers the PRAM CREW (Parallel random-access machine with Concurrent Read, Exclusive Write) model as considered in [28]. Thus, the same memory location can be read by multiple processors simultaneously. However, the same memory location cannot be written at the same time by multiple processors. As simultaneous write operations are not allowed, so there will be no concurrent write operation in our parallel version. The analysis of parallel algorithms is usually carried out under the assumption that an unbounded number of processors is available [15, 22]. So, in our analysis we have also considered the same assumption. In the analysis of the parallel approach, we obtain the maximum number of processors which can be required.

4 The DCNS Approach and its Parallelization

In this section, we discuss the DCNS approach and we also explore the scope of parallelism within this approach. DCNS is a two-phased approach. In the first phase, the solutions are sorted based on the first objective as in [7, 9, 11, 32, 33]. After this sorting, the solution which comes later in the sorted list will never dominate the former solutions [32]. So, when two solutions are compared, there are only two possibilities. In the second phase, the actual sorting is performed.

In the second phase, initially, each solution in the sorted list of solutions is considered as a set of fronts, *i.e.*, initially, each set of fronts has a single front which has a single solution. So, initially we get N sets of fronts corresponding to N solutions. The approach is based on a divide-and-conquer strategy. So, these sets of fronts are merged at $\mathcal{L} = \log N$ different levels like in merge sort. However, the merging will be different than in the normal merge sort. At the first level, $N/2$ merge operations are performed where $N/2$ pairs of the sets of fronts are merged. At the second level, $N/4$ merge operations are performed where $N/4$ pairs of the sets of fronts are merged. In general, at the l^{th} level, $N/2^l$ merge operations are performed.

Let the merge operation be performed between two sets of fronts $\mathcal{F}$ and $\mathcal{F}'$ where $\mathcal{F} = \{F_1, F_2, \ldots, F_P\}$ and $\mathcal{F}' = \{F'_1, F'_2, \ldots, F'_Q\}$. In the merge operation, all the solutions from $\mathcal{F}'$ are inserted into their respective position in $\mathcal{F}$. At each moment, all the solutions in a set of fronts are arranged in decreasing order of their dominance. Also, the solutions of the $(k+1)^{th}$ set of fronts cannot dominate the solutions of the k^{th} set of fronts. This is because, initially, different sets of fronts are created from the sorted solutions and in the sorted list, a solution which comes later cannot dominate the previous solutions. So, when two sets of fronts $\mathcal{F}$ and $\mathcal{F}'$ are merged, then the solutions of $\mathcal{F}'$ cannot dominate the solutions in $\mathcal{F}$.

The working flow of the DCNS framework is shown in Fig. 1. Let's assume that we have eight solutions $\{sol_1, sol_2, \ldots, sol_8\}$. Initially, there are eight sets of fronts $\{\mathcal{F}_1, \mathcal{F}_2, \ldots, \mathcal{F}_8\}$ corresponding to eight sorted solutions in the first phase. At the first level, four merge operations are performed. The sets of

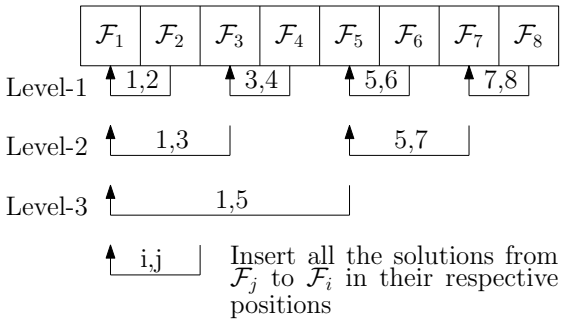

Fig. 1 Flow of the DCNS Framework

fronts at index positions 1 and 2 are merged. Similarly, the set of fronts at index positions 3 and 4 are merged. The set of fronts at index positions 5 and 6 are merged and also the set of fronts at index positions 7 and 8 are merged. At the second level, two merge operations are performed. The sets of fronts at index positions 1 and 3 are merged and the set of fronts at index positions 5 and 7 are merged. At the third level, only one merge operation is performed. The sets of fronts at index positions 1 and 5 are merged to get the final sorted set of fronts.

The merge operation is shown in Fig. 2. In this operation, all the solutions from $\mathcal{F}'$ are inserted into their respective position in $\mathcal{F}$. When the solutions from $\mathcal{F}'$ are inserted in $\mathcal{F}$, then initially the solutions of F'_1 are inserted, then the solutions of F'_2 are inserted and so on. When the solutions of a front $F' \in \mathcal{F}'$ are inserted into $\mathcal{F}$, the three dominance relationships are used as discussed in [17] to avoid unnecessary dominance comparisons. There are two ways to insert a solution in $\mathcal{F}$: sequential search based insertion and binary search based insertion as in [32], [17]. Depending on the type of insertion, there are two approaches: DCNS-SS (DCNS based on sequential search) and DCNS-BS (DCNS based on binary search).

In general, the recurrence relation of the second phase of the DCNS approach is defined by Eq. (1). As the approach is based on a divide-and-conquer strategy, so the problem of size N is divided into two equal sized sub-problems which are then solved. $T_{\text{dom}}(N, M)$ represents the time spent to perform the dominance comparisons when the solutions from $\mathcal{F}'$ are inserted into $\mathcal{F}$ while solving the problem of size N where both sets of fronts have $N/2$ solutions. $T_{\text{insert}}(N)$ represents the time to add $N/2$ solutions in $\mathcal{F}$ from $\mathcal{F}'$ while solving the problem of size N.

$$T_1(N, M) = \begin{cases} M+1 & \text{if } N=2 \\ 2T_1\left(N/2, M\right) + T_{\text{dom}}(N, M) + T_{\text{insert}}(N) & \text{otherwise} \end{cases} \quad (1)$$

In general, the value of $T_{dom}(N, M)$ changes depending on the dominance relation between the solutions. In a merge operation where $\mathcal{F}$ and $\mathcal{F}'$ have $N/2$ solu-

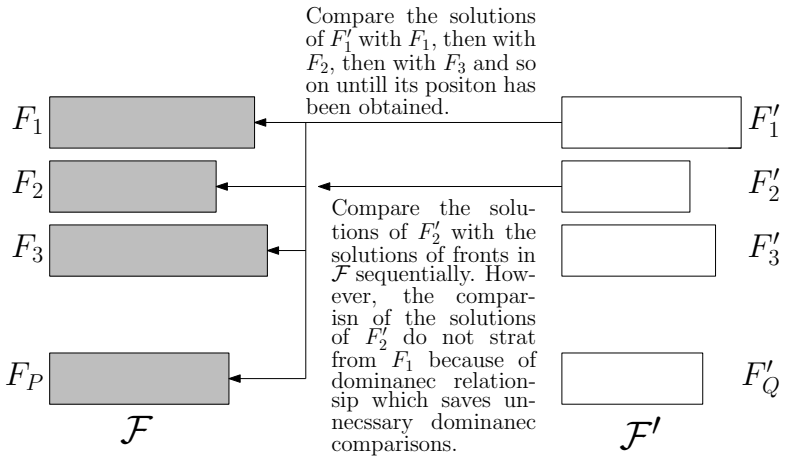

Fig. 2 Working of the Merge procedure

tions, all the solutions from $\mathcal{F}'$ need to be inserted into $\mathcal{F}$ so, $T_{\text{insert}}(N) = {}^{N}/_{2}$.

Parallelism in the First Phase: In the first phase, the solutions are sorted based on the first objective. For sorting the solutions, parallel merge sort can be used where different merge sort procedures can be performed simultaneously. Using parallel merge sort, the worst case time complexity of the first phase is $\mathcal{O}(MN)$ and the best case time complexity is $\mathcal{O}(N)$. To further improve the performance of the first phase, the merge operation in parallel merge sort can also be performed in a parallel manner as discussed in [3]. By doing this, the best case time complexity of the first phase will become $\mathcal{O}(\log^3 N)$. Merge sort is used in the first phase, so the space complexity of the first phase is $\mathcal{O}(N)$.

In parallel merge sort, different merge operations at the same level can be performed simultaneously. So, the maximum number of processors required to perform the merge operation at the first level is ${}^{N}/_{2}$, at the second level it is ${}^{N}/_{4}$ and so on. At the last level, the number of processors required is one. To further speed up parallel merge sort, the merge operation can itself be implemented in a parallel manner. The maximum number of processors required to perform a merge operation where the size of both arrays is ${}^{N}/_{2}$, is N. So, the maximum number of processors required to perform the merge operation at each level is N when the merge operation is itself implemented in a parallel manner. Thus, the maximum number of processors required to perform parallel merge sort is N.

Parallelism in Dominance Comparisons: In general, the dominance relation between two solutions can be obtained in $\mathcal{O}(M)$ time. The dominance relation between each pair of solutions can be obtained in parallel, so the dominance matrix which contains the dominance relation between each pair of solutions can be obtained in $\mathcal{O}(M)$ time in parallel. The dominance matrix in this manner is obtained in [28]. However, this $\mathcal{O}(M)$ time complexity can

be further improved if the dominance relation between a pair of solutions can be obtained in less time than $\mathcal{O}(M)$.

To compare two solutions sol_i and sol_j, we create two Boolean arrays of size M. The first array (say D_i) stores whether sol_i is better than sol_j for each of the M objectives. The second array (say D_j) stores whether sol_j is better than sol_i for each of 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 D_i is set to 'TRUE'; otherwise, it is set to 'FALSE'. Similarly, the Boolean array D_j is also filled.

We process arrays D_i and D_j simultaneously. These arrays are processed at $\log M$ levels. At each level, we perform an 'OR' operation between two consecutive array cells. At the l^{th} level, $M/2^l$ 'OR' operations are performed. After the 'OR' operation at the last level, either a 'TRUE' or 'FALSE' value is obtained. So, we get two Boolean values corresponding to both arrays. Let the final value obtained after processing D_i be denoted by D_{ifinal} and the final value obtained after processing D_j be denoted by D_{jfinal}. With the help of these two values D_{ifinal} and D_{jfinal}, the dominance relationship between the solutions is obtained as follows.

1. $\boldsymbol{D_{\text{ifinal}} = D_{\text{jfinal}} =}$ **False:** Solutions sol_i and sol_j are the same (in terms of objective values).
2. $\boldsymbol{D_{\text{ifinal}} = D_{\text{jfinal}} =}$ **True:** Solutions sol_i and sol_j are non-dominated.
3. $\boldsymbol{D_{\text{ifinal}} =}$ **True and** $\boldsymbol{D_{\text{jfinal}} =}$ **False:** Solution sol_i dominates sol_j.
4. $\boldsymbol{D_{\text{ifinal}} =}$ **False and** $\boldsymbol{D_{\text{jfinal}} =}$ **True:** Solution sol_i is dominated by sol_j.

Both Boolean arrays D_i and D_j can be filled in $\mathcal{O}(1)$ time in parallel. As different 'OR' operations at the same level can be performed simultaneously in these two Boolean arrays, so the time complexity of processing these arrays in a parallel manner is $\mathcal{O}(\log M)$. Thus, the dominance relationship between a pair of solutions can be obtained in $\mathcal{O}(\log M)$ time. Hence, the dominance matrix can be obtained in $\mathcal{O}(\log M)$ time in parallel as the dominance relation between each pair of solutions can be obtained simultaneously. After obtaining the dominance matrix, it requires $\mathcal{O}(1)$ time to find whether a solution is dominated by another solution or not, as only a lookup in the dominance matrix is required.

The maximum number of processors required to fill Boolean array D_i is M and the maximum number of processors required to fill Boolean array D_j is also M. Both these Boolean arrays are filled simultaneously, so the number of processors required to fill both Boolean arrays is $2M$. These two Boolean arrays are processed simultaneously at $\log M$ levels. The number of processors required to process a Boolean array at the first level is $M/2$, at the second level it is $M/4$ and so on. At the last level, the number of processors required is one. So, the number of processors required to process both Boolean arrays simultaneously is $M/2+M/2 = M$. Thus, the total number of processors required to obtain the dominance relationship between two solutions is $2M$ which is $\mathcal{O}(M)$. The dominance relationship between each pair of solutions can be obtained simultaneously. There are N^2 pairs of solutions so the maximum

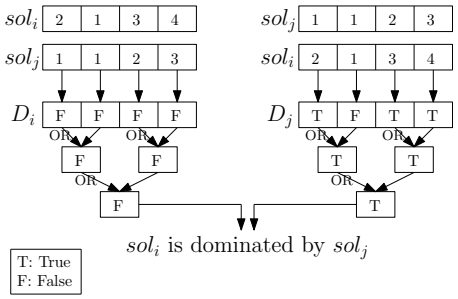

Fig. 3 The process to obtain the dominance relationship between the two solutions sol_i and sol_j in a parallel manner.

number of processors required to obtain the dominance matrix simultaneously is $2MN^2$ or $\mathcal{O}(MN^2)$.

Example 1 *Let's assume that we want to find the dominance relationship between two solutions* $sol_i = \{2, 1, 3, 4\}$ *and* $sol_j = \{1, 1, 2, 3\}$ *which are in* 4*-dimensional space. For this purpose, we compare* sol_i *with* sol_j *to fill array* D_i *and compare* sol_j *with* sol_i *to fill array* D_j*. These two arrays are processed in a parallel manner using 'OR'. After processing* D_i*, we are getting '*False*' and after processing* D_j*, we are getting '*True*' which means that* sol_i *is dominated by* sol_j*. The complete process to obtain the dominance relationship between the two solutions is shown in Fig. 3.*

Parallelism in the Second Phase: To obtain the parallelism in the second phase, all the merge operations at the same level can be performed simultaneously. Along with this, the position of different solutions of a front $F' \in \mathcal{F}'$ are identified in $\mathcal{F}$ simultaneously. In the process of identifying the position of a solution $sol' \in F'$ in $\mathcal{F}$, the solution sol' is compared with respect to all the solutions of a front $F \in \mathcal{F}$ simultaneously. After obtaining the position of each solution of F' in $\mathcal{F}$, these solutions are added to their corresponding front in $\mathcal{F}$ in a sequential manner to avoid write collisions as we are considering the PRAM CREW model.

Let the number of solutions in a front $F \in \mathcal{F}$ be n_f. A particular solution $sol' \in F'$ will be compared with all n_f solutions of F simultaneously. The dominance relation of sol' with respect to all the solutions of F is stored in an array named as dominance array of size n_f. 'True' in this array denotes that sol' is dominated by a solution of F and 'False' denotes that sol' is non-dominated with respect to a solution of F. The time to obtain the dominance relation of sol' with respect to all the solutions of F in a parallel manner is $\mathcal{O}(1)$ as only the lookup in the dominance matrix is required.

Now, we check whether sol' is non-dominated with respect to all the solutions of F or not. For this purpose, the dominance array is processed in a parallel manner using an 'AND' operation. As the size of the array is n_f, the array is processed in a parallel manner at $\log n_f$ levels. At each level, the consecutive array cells are processed using an 'AND' operation. So, at the l^{th} level

$n_f/2^i$ 'AND' operations are performed. Thus, the time complexity of processing this array in a parallel manner is $\mathcal{O}(\log n_f)$. Thus, the overall time complexity to compare sol' with respect to all the solutions of F and knowing whether sol' is non-dominated with respect to all the solutions of F in a parallel manner is $\mathcal{O}(1 + \log n_f)$. As all the solutions of F' are simultaneously compared with respect to the solutions of F, so the time to obtain the dominance relation of all the solutions of F' with respect to all the solutions of F is also $\mathcal{O}(1 + \log n_f)$.

At the i^{th} level, there are $N/2^i$ merge operations. The number of solutions in the set of fronts which take part in the merge procedure at the i^{th} level is 2^{i-1}. In the parallel merge procedure, the position of different solutions of a front $F' \in \mathcal{F}'$ is identified in $\mathcal{F}$ simultaneously. As the maximum number of solutions in a front $F' \in \mathcal{F}'$ at i^{th} level can be 2^{i-1} (which occurs when all the solutions are in the same front in $\mathcal{F}'$), so the maximum number of processors required to find the position of each solution of $\mathcal{F}'$ in $\mathcal{F}$ is 2^{i-1}. Also, in the process of identifying the position of a particular solution $sol' \in F'$ in $\mathcal{F}$, the solution sol' is compared with respect to all the solutions of a front $F \in \mathcal{F}$ simultaneously. The maximum number of solutions in a front $F \in \mathcal{F}$ can also be 2^{i-1} (which occurs when all the solutions are in a single front in $\mathcal{F}$), so the maximum number of processors required to find the position of a solution sol' in $\mathcal{F}$ is also 2^{i-1}. Thus, the maximum number of processors required to merge two set of fronts at the i^{th} level is $2^{i-1} \cdot 2^{i-1} = 2^{2i-2}$. Thus, the maximum number of processors required at the i^{th} level is given by Eq. (2).

$$\text{Maximum number of processors} = \left(\frac{N}{2^i}\right)\left(2^{2i-2}\right) = \frac{1}{4}N2^i \tag{2}$$

The maximum value of Eq. (2) can be obtained when the value of 2^i is maximum and this occurs at the last level. Thus, the maximum value of Eq. (2) is $1/4N^2$. Thus, the maximum number of processors required is $N^2/4$.

The maximum number of processors required to sort the solutions based on a particular objective using parallel merge sort is N. The maximum number of processors required to obtain the dominance matrix simultaneously is $2MN^2$. The maximum number of processors required to perform different merge operations is $N^2/4$. Thus, the overall maximum number of processors required to perform DCNS approach in parallel is $2MN^2$ or $\mathcal{O}(MN^2)$.

Example 2 *Let's assume that two sets of fronts $\mathcal{F}$ and $\mathcal{F}'$ need to be merged. Consider a front $F \in \mathcal{F}$ and $F' \in \mathcal{F}'$ where $F = \{s_1, s_2, s_3, s_4\}$ and $F' = \{s_5, s_6, s_7, s_8\}$. These two fronts are shown in Fig. 4. To obtain the parallelism, all the solutions in F' are simultaneously compared with all the solutions in F. Solutions s_5, s_6, s_7, s_8 are compared simultaneously. Each of these solutions are also compared with respect to four solutions in F simultaneously. Fig. 4 shows the parallel comparisons of all the solutions of F' with all the solutions of F.*

Now, we discuss the parallelism in the second phase in the following four scenarios:

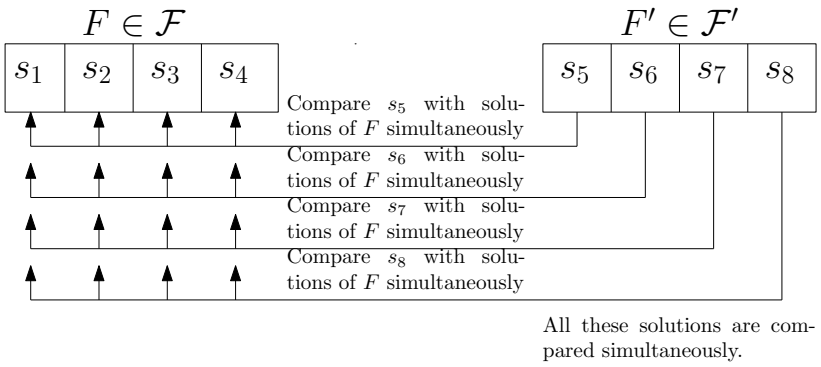

Fig. 4 Parallelism in the DCNS based approach.

1. All the solutions are in a single front.
2. All the solutions are in different fronts.
3. N solutions are equally divided in $\sqrt{N}$ fronts such that each solution in a front is dominated by all the solutions in its preceding front.
4. N solutions are equally divided in $\sqrt{N}$ fronts such that each solution in a front is dominated by only one solution in its preceding front.

We establish the recurrence relation for the serial and parallel version of the second phase. As the approach is based on a divide-and-conquer strategy, we are assuming N to be a perfect square.

5 Scenario – I

The number of fronts is one, so both the sequential and the binary search based approaches perform the same. As all the solutions are non-dominated so there will be only a single front in both sets of fronts $\mathcal{F}$ and $\mathcal{F}'$ in the merge operation.

5.1 Serial Approach

The recurrence relation of the serial version is given by Eq. (3). In a merge operation where both sets of fronts have $N/2$ solutions, each solution of $\mathcal{F}'$ is compared with respect to all $N/2$ solutions in $\mathcal{F}$. Thus, $T_{\text{dom}}(N, M) = M\,(N/2)^2$.

$$T_1(N, M) = \begin{cases} M+1 & \text{if } N=2 \\ 2T_1\,(N/2, M) + M\,(N/2)^2 + N/2 & \text{otherwise} \end{cases} \tag{3}$$

The recurrence relation in Eq. (3) is solved by Eq. (4).

$$\begin{aligned} T_1(N, M) &= \left[M\,(N/2)^2 + N/2\right] + 2\left[M\,(N/4)^2 + N/4\right] + \ldots + N/2\left[M\,(N/N)^2 + N/N\right] \\ &= 1/2MN(N-1) + 1/2N\log N \end{aligned} \tag{4}$$

5.2 Parallel Approach

In the parallel version, all the merge operations are performed simultaneously. Also, the position of each solution in $\mathcal{F}'$ is identified in $\mathcal{F}$ simultaneously. In a merge operation where both sets of fronts have $N/2$ solutions, the time complexity to know whether a particular solution of $\mathcal{F}'$ is non-dominated with respect to all the solutions of $\mathcal{F}$ is $\mathcal{O}(1 + \log(N/2))$. As all the solutions of $\mathcal{F}'$ are simultaneously compared with all the solutions of $\mathcal{F}$ so the time to obtain the dominance relation of all the solutions of $\mathcal{F}'$ with all the solutions of $\mathcal{F}$ is also $\mathcal{O}(1 + \log(N/2))$. Thus, $T_{\text{dom}}(N, M) = 1 + \log(N/2)$. The recurrence relation of the parallel version is given by Eq. (5).

$$T_\infty(N) = \begin{cases} 1+1 & \text{if } N=2 \\ T_\infty(N/2, M) + 1 + \log(N/2) + N/2 & \text{otherwise} \end{cases} \tag{5}$$

The recurrence relation in Eq. (5) is solved by Eq. (6).

$$\begin{aligned} T_\infty(N) &= [1 + \log(N/2) + N/2] + [1 + \log(N/4) + N/4] + \ldots + [1 + \log(N/N) + N/N] \\ &= 1/2(2N + \log^2 N + \log N - 2) \end{aligned} \tag{6}$$

6 Scenario – II

In this scenario, the number of fronts is more than one, so sequential and binary search based approaches perform differently. In this scenario, as all the solutions are in different fronts, so in a merge operation where both sets of fronts have $N/2$ fronts which contain a single solution, only the solution in the first front in $\mathcal{F}'$ is compared and dominated by the solutions of all the fronts in $\mathcal{F}$. The solutions of the remaining fronts in $\mathcal{F}'$ are directly added to $\mathcal{F}$ without performing dominance comparisons because of the dominance relationship as discussed in [17].

In case of the parallel version of this scenario, only different merge operations can be performed simultaneously. As each front has a single solution, so a solution of front $F' \in \mathcal{F}$ is compared with a solution of $F \in \mathcal{F}$ in $\mathcal{O}(1)$ time.

6.1 Sequential Search based Approach

Here, the recurrence relation is established using a sequential search based strategy.

6.1.1 Serial Approach

The recurrence relation of the serial version is given by Eq. (7). In a merge operation where both sets of fronts have $N/2$ solutions, only the solution of

the first front in $\mathcal{F}'$ is compared with the solutions of all the fronts in $\mathcal{F}$, so $T_{\text{dom}}(N, M) = M\,(N/2)$.

$$T_1(N, M) = \begin{cases} M+1 & \text{if } N=2 \\ 2T_1\,(N/2, M) + M\,(N/2) + N/2 & \text{otherwise} \end{cases} \tag{7}$$

The recurrence relation in Eq. (7) is solved by Eq. (8).

$$\begin{aligned} T_1(N, M) &= [M\,(N/2) + N/2] + 2\,[M\,(N/4) + N/4] + \ldots + N/2\,[M\,(N/N) + N/N] \\ &= 1/2 MN \log N + 1/2 N \log N \end{aligned} \tag{8}$$

6.1.2 Parallel Approach

In the parallel version, all the merge operations are performed simultaneously. So, the recurrence relation of the parallel version is given by Eq. (9). Here, only the solution of the first front in $\mathcal{F}'$ is compared with the solutions of all the fronts in $\mathcal{F}$, so $T_{\text{dom}}(N) = (N/2)$.

$$T_\infty(N) = \begin{cases} 1+1 & \text{if } N=2 \\ T_\infty\,(N/2, M) + (N/2) + N/2 & \text{otherwise} \end{cases} \tag{9}$$

The recurrence relation in Eq. (9) is solved by Eq. (10).

$$\begin{aligned} T_\infty(N) &= [(N/2) + N/2] + [(N/4) + N/4] + \ldots + [(N/N) + N/N] \\ &= 2(N-1) \end{aligned} \tag{10}$$

6.2 Binary Search based Approach

Now, we obtain the recurrence relation using a binary search based strategy for both the serial and the parallel version.

6.2.1 Serial Approach

The recurrence relation of the serial version is given by Eq. (11). Here, only the solution of the first front in $\mathcal{F}'$ is compared with the solutions of $\lceil \log (N/2+1) \rceil$ fronts in $\mathcal{F}$, so $T_{\text{dom}}(N, M) = M \lceil \log (N/2+1) \rceil$.

$$T_1(N, M) = \begin{cases} M+1 & \text{if } N=2 \\ 2T_1\,(N/2, M) + M \lceil \log (N/2+1) \rceil + N/2 & \text{otherwise} \end{cases} \tag{11}$$

The recurrence relation in Eq. (11) is solved by Eq. (12).

$$\begin{aligned} T_1(N, M) &= \left[M \lceil \log (N/2+1) \rceil + N/2\right] + 2\left[M \lceil \log (N/4+1) \rceil + N/4\right] + \ldots \\ &\quad + N/2\left[M \lceil \log (N/N+1) \rceil + N/N\right] \\ &= M(2N - \log N - 2) + 1/2 N \log N \end{aligned} \tag{12}$$

6.2.2 Parallel Approach

In the parallel version, all the merge operations are performed simultaneously. So, the recurrence relation of the parallel version is given by Eq. (13). Here, only the solution of the first front in $\mathcal{F}'$ is compared with the solutions of $\lceil \log(N/2+1) \rceil$ fronts in $\mathcal{F}$, so $T_{\text{dom}}(N) = \lceil \log(N/2+1) \rceil$.

$$T_\infty(N) = \begin{cases} 1+1 & \text{if } N=2 \\ T_\infty(N/2, M) + \lceil \log(N/2+1) \rceil + N/2 & \text{otherwise} \end{cases} \tag{13}$$

The recurrence relation in Eq. (13) is solved using Eq. (14).

$$\begin{aligned} T_\infty(N) &= \left[\lceil \log(N/2+1) \rceil + N/2 \right] + \left[\lceil \log(N/4+1) \rceil + N/4 \right] + \dots \\ &\quad + \left[\lceil \log(N/N+1) \rceil + N/N \right] \\ &= 1/2(2N + \log^2 N + \log N - 2) \end{aligned} \tag{14}$$

7 Scenario – III

We establish the recurrence relation for the serial and the parallel version when N solutions are equally divided into $\sqrt{N}$ fronts such that each solution in a front is dominated by all the solutions in its preceding front. As the number of fronts is more than one, so the sequential and the binary search based approaches perform differently. Let us consider $N' = \sqrt{N}$.

7.1 Sequential Search based Approach

Now, we obtain the recurrence relation using a sequential search based strategy for both the serial and the parallel version.

7.1.1 Serial Approach

The recurrence relation of the serial version is given by Eq. (15). This recurrence relation is in two parts. The first part corresponds to the merge operation from the first to the $\mathcal{L}/2^{th}$ level and the second part corresponds to the merge operation from the $\mathcal{L}/2 + 1^{th}$ to the last level.

$$T_1(N, M) = N' T_{11}(\sqrt{N}, M) + \underbrace{T_{12}(N, M)}_{N > N'} \tag{15}$$

First Part of Recurrence Relation: In the final sorted fronts, each front has $\sqrt{N} (= 2^{\mathcal{L}/2})$ solutions. So, in the merge operations from the first to the $\mathcal{L}/2^{th}$ level, all the solutions are non-dominated. As all the solutions are non-dominated till the $\mathcal{L}/2^{th}$ level, there will be only one single front in both the set of fronts $\mathcal{F}$ and $\mathcal{F}'$ in the merge operations till the $\mathcal{L}/2^{th}$ level. Thus, in

a merge operation where both sets of fronts have $\sqrt{N}/2$ solutions, each solution of $\mathcal{F}'$ is compared with respect to all the $\sqrt{N}/2$ solutions in $\mathcal{F}$. Hence, $T_{\text{dom}}(\sqrt{N}, M) = M\left(\sqrt{N}/2\right)^2$. All the solutions from $\mathcal{F}'$ need to be inserted into $\mathcal{F}$. Thus, $T_{\text{insert}}(\sqrt{N}) = \sqrt{N}/2$. So, the recurrence relation corresponding to the first part is given by Eq. (16) which is solved using Eq. (17).

$$T_{11}(\sqrt{N}, M) = \begin{cases} M+1 & \text{if } \sqrt{N}=2 \\ 2T_{11}\left(\frac{\sqrt{N}}{2}, M\right) + M\left(\frac{\sqrt{N}}{2}\right)^2 + \frac{\sqrt{N}}{2} & \text{otherwise} \end{cases} \tag{16}$$

$$\begin{aligned} T_{11}(\sqrt{N}, M) &= \left[M\left(\sqrt{N}/2\right)^2 + \sqrt{N}/2\right] + 2\left[M\left(\sqrt{N}/4\right)^2 + \sqrt{N}/4\right] + \ldots \\ &\quad + \sqrt{N}/2\left[M\left(\sqrt{N}/\sqrt{N}\right)^2 + \sqrt{N}/\sqrt{N}\right] \\ &= 1/2 M\sqrt{N}(\sqrt{N}-1) + 1/4\sqrt{N}\log N \end{aligned} \tag{17}$$

Second Part of the Recurrence Relation: At the end of the merge operation at the $\mathcal{L}/2^{th}$ level, each set of fronts has a single front which has $\sqrt{N}$ solutions. After the merge operation at the $\mathcal{L}/2^{th}$ level, each set of fronts has more than one front and each front has $\sqrt{N}$ solutions. Whenever a merge operation is performed after the $\mathcal{L}/2^{th}$ level, only the solutions of the first front in $\mathcal{F}'$ are compared with the solutions of $\mathcal{F}$. The solutions of the remaining fronts in $\mathcal{F}'$ are added directly to $\mathcal{F}$ because of the dominance relationship as discussed in [17]. The solutions of the front $F' \in \mathcal{F}'$ are only compared and dominated by a single solution in all the fronts in $\mathcal{F}$ because each solution in a front is dominated by all the solutions in its preceding front.

In a merge operation where both sets of fronts have $N/2$ solutions, only $\sqrt{N}$ solutions in the first front in $\mathcal{F}'$ are compared with the first solution in all $N/2\sqrt{N}$ fronts in $\mathcal{F}$. So, $T_{\text{dom}}(N, M) = M\left(N/2\sqrt{N}\right)\sqrt{N} = M\left(N/2\right)$. All the solutions from $\mathcal{F}'$ need to be inserted into $\mathcal{F}$ so, $T_{\text{insert}}(N) = N/2$. Thus, the recurrence relation corresponding to the second part is given by Eq. (18) which is solved using Eq. (19).

$$\begin{aligned} \underbrace{T_{12}(N, M)}_{N>N'} &= \begin{cases} MN'+N' & \text{if } N=2N' \\ 2T_{12}\left(\frac{N}{2}, M\right) + M\left(\frac{N}{2N'}\right)N' + \left(\frac{N}{2N'}\right)N' & \text{otherwise} \end{cases} \\ &= \begin{cases} MN'+N' & \text{if } N=2N' \\ 2T_{12}\left(\frac{N}{2}, M\right) + M\left(\frac{N}{2}\right) + \frac{N}{2} & \text{otherwise} \end{cases} \end{aligned} \tag{18}$$

$$\begin{aligned} T_{12}(N, M) &= \left[M\left(N/2\right) + N/2\right] + 2\left[M\left(N/4\right) + N/4\right] + \ldots + N'/2\left[M\left(N/N'\right) + N/N'\right] \\ &= 1/4 MN\log N + 1/4 N\log N \end{aligned} \tag{19}$$

The solution to the recurrence relation in Eq. (15) is obtained by Eq. (20).

$$\begin{aligned} T_1(N, M) &= \sqrt{N}\left[1/2 M\sqrt{N}(\sqrt{N}-1) + 1/4\sqrt{N}\log N\right] + 1/4 MN\log N + 1/4 N\log N \\ &= 1/4 MN(2\sqrt{N} + \log N - 2) + 1/2 N\log N \end{aligned} \tag{20}$$

Consider Fig. 5 which shows the working of the DCNS approach when sixteen solutions are equally divided into four fronts such that each solution in a front is dominated by all the solutions in its preceding front. As the number of solutions is 16, so the merge operations are performed in $\mathcal{L} = \log 16 = 4$ levels. Till the merge operations at the second level, all the solutions are non-dominated, so the recurrence relation $T_{11}(\sqrt{N}, M)$ is used four times since the initial input to this recurrence relation is four and there are a total of sixteen solutions. After the merge operation at the second level, the recurrence relation $T_{12}(N, M)$ is used.

From this figure, it is clear that at the end of the merge operation at the second level, each set of fronts has a single front which has $4(=\sqrt{16})$ solutions. After the merge operation at the second level, each set of fronts has more than one front and each front has $4(=\sqrt{16})$ solutions. When the merge operation is performed at the last level where $\mathcal{F}$ and $\mathcal{F}'$ both have two fronts, s_9 is compared and dominated by the first solution of F_1, *i.e.*, with s_1. After this, s_9 is compared with the solution of F_2. Now, s_9 is compared and dominated by the first solution of F_2, *i.e.*, with s_5. So, s_9 will make a new front. Similarly, s_{10}, s_{11} and s_{12} are also compared and dominated by the first solution of F_1 and F_2, *i.e.*, with s_1 and s_5. After that, these three solutions s_{10}, s_{11} and s_{12} will be inserted into the new front where s_9 has been inserted.

Once all the solutions of the first front in $\mathcal{F}'$ are inserted into $\mathcal{F}$, then all the solutions of the second front in $\mathcal{F}'$ are directly inserted into a new front in $\mathcal{F}$ because of the dominance relationship as discussed in [17].

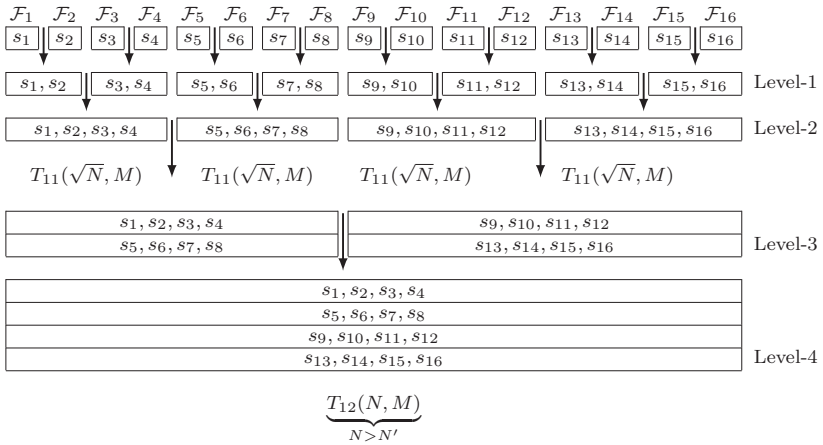

Fig. 5 Working procedure to sort sixteen solutions when those are equally divided into four fronts and each solution in a front is dominated by all the solutions in its preceding front. Down arrow indicates the merge operation between immediate left and right set of fronts. $s_i, \ldots, s_j$ represent that these solutions are non-dominated with respect to each other. Here, up to the 2^{nd} level, the recurrence relation $\sqrt{N}T_{11}(\sqrt{N}, M)$ is used and from the 3^{rd} to the last level, the recurrence relation $T_{12}(N, M)$ is used .

7.1.2 Parallel Approach

The recurrence relation of the parallel version is given by Eq. (21).

$$T_\infty(N) = N' T_{\infty 1}(\sqrt{N}) + \underbrace{T_{\infty 2}(N)}_{N>N'} \tag{21}$$

Till the $\mathcal{L}/2^{th}$ level, all the solutions are non-dominated. In the parallel version, the position of a different solution of a front $F' \in \mathcal{F}'$ is identified in $\mathcal{F}$ simultaneously. In a merge operation where both sets of fronts have $\sqrt{N}/2$ solutions, the time complexity to know whether a particular solution of $\mathcal{F}'$ is non-dominated with respect to all the solutions of $\mathcal{F}$ is $\mathcal{O}(1+\log\left(\sqrt{N}/2\right))$. As all the solutions of $\mathcal{F}'$ are simultaneously compared with the solutions of $\mathcal{F}$ so the time to obtain the dominance relation of all the solutions of $\mathcal{F}'$ with respect to all the solutions of $\mathcal{F}$ is also $\mathcal{O}(1+\log\left(\sqrt{N}/2\right))$. Thus, $T_{\text{dom}}(\sqrt{N}) = 1+\log\left(\sqrt{N}/2\right)$. All the solutions from $\mathcal{F}'$ need to be inserted in $\mathcal{F}$ so $T_{\text{insert}}(\sqrt{N}) = \sqrt{N}/2$. The recurrence relation corresponding to the first part is given by Eq. (22) which is solved using Eq. (23).

$$T_{\infty 1}(\sqrt{N}) = \begin{cases} 1+1 & \text{if } \sqrt{N}=2 \\ T_{\infty 1}\left(\frac{\sqrt{N}}{2}\right) + 1 + \log\left(\frac{\sqrt{N}}{2}\right) + \frac{\sqrt{N}}{2} & \text{otherwise} \end{cases} \tag{22}$$

$$\begin{aligned} T_{\infty 1}(\sqrt{N}) &= \left[1+\log\left(\sqrt{N}/2\right)+\sqrt{N}/2\right] + \left[1+\log\left(\sqrt{N}/4\right)+\sqrt{N}/4\right] + \ldots \\ &\quad + \left[1+\log\left(\sqrt{N}/\sqrt{N}\right)+\sqrt{N}/\sqrt{N}\right] \\ &= 1/8\left(8\sqrt{N} + \log^2 N + 2\log N - 8\right) \end{aligned} \tag{23}$$

After the $\mathcal{L}/2^{th}$ level, each front has $\sqrt{N}$ solutions. Whenever a merge operation is performed after the $\mathcal{L}/2^{th}$ level, only the solutions of the first front in $\mathcal{F}'$ are compared with the solutions of $\mathcal{F}$. The solutions of the remaining fronts in $\mathcal{F}'$ are added directly to $\mathcal{F}$ because of the dominance relationship as discussed in [17].

In a merge operation where both sets of fronts have $N/2$ solutions, only $\sqrt{N}$ solutions in the first front in $\mathcal{F}'$ are compared with respect to all the solutions of front $F \in \mathcal{F}$ simultaneously. The time complexity of obtaining the dominance relation of a solution of the front in $\mathcal{F}'$ with respect to all the solutions of a front $F \in \mathcal{F}$ is $\mathcal{O}(1+\log N')$. As all the solutions of a front $F' \in \mathcal{F}'$ are simultaneously compared with the solutions of a front $F \in \mathcal{F}$, so the time to obtain the dominance relation of all the solutions of a front $F' \in \mathcal{F}'$ with respect to all the solutions of a front $F \in \mathcal{F}$ is also $\mathcal{O}(1+\log N)$. Thus, $T_{\text{dom}}(N) = (N/2N')\,(1+\log N')$. All the solutions from $\mathcal{F}'$ need to be inserted in $\mathcal{F}$ so $T_{\text{insert}}(N) = N/2$. Hence, the recurrence relation corresponding to the

second part is given by Eq. (24) which is solved using Eq. (25).

$$\underbrace{T_{\infty 2}(N)}_{N>N'}=\begin{cases}1+\log N'+N' & \text{if } N=2N'\\ T_{\infty 2}\left(\frac{N}{2}\right)+\frac{N}{2N'}\left(1+\log N'\right)+\left(\frac{N}{2N'}\right)N' & \text{otherwise}\end{cases}$$
$$=\begin{cases}1+\log N'+N' & \text{if } N=2N'\\ T_{\infty 2}\left(\frac{N}{2}\right)+\frac{N}{2N'}\left(1+\log N'\right)+\frac{N}{2} & \text{otherwise}\end{cases} \quad (24)$$

$$\begin{aligned}T_{\infty 2}(N)=&\left[N/2N'\left(1+\log N'\right)+N/2\right]+\left[N/4N'\left(1+\log N'\right)+N/4\right]+\\ &\ldots+\left[N/N'N'\left(1+\log N'\right)+N/N'\right]\\ =&1/2(2N+\sqrt{N}\log N-\log N-2)\end{aligned} \quad (25)$$

The solution to the recurrence relation in Eq. (21) is obtained by Eq. (26).

$$\begin{aligned}T_{\infty}(N)=&\sqrt{N}\left[1/8(8\sqrt{N}+\log^2 N+2\log N-8)\right]+1/2(2N+\sqrt{N}\log N-\log N-2)\\ =&2N+1/8\sqrt{N}\log^2 N+3/4\sqrt{N}\log N-\sqrt{N}-1/2\log N-1\end{aligned} \quad (26)$$

7.2 Binary Search based Approach

Here, the recurrence relation is established using a binary search based approach. In the binary search based approach, the merge operations up to the $\mathcal{L}/2^{th}$ level remain the same as in the sequential search based approach because up to the $\mathcal{L}/2^{th}$ level all the solutions are in a single front. After the $\mathcal{L}/2^{th}$ level, in a merge operation where both sets of fronts have $N/2$ solutions, $\sqrt{N}$ solutions in the first front in $\mathcal{F}'$ are compared with the first solution in $\lceil \log\left(N/2\sqrt{N}+1\right)\rceil$ fronts in $\mathcal{F}$.

7.2.1 Serial Approach

The recurrence relation of the serial version is given by Eq. (15). The first part of the recurrence relation, *i.e.*, $T_{11}(\sqrt{N},M)$ is the same as Eq. (16) so the solution to Eq. (16) is obtained by Eq (17). The second part of the recurrence relation is given by Eq. (27) which is solved using Eq. (28).

$$\underbrace{T_{12}(N,M)}_{N>N'}=\begin{cases}MN'+N' & \text{if } N=2N'\\ 2T_{12}\left(\frac{N}{2},M\right)+M\left\lceil\log\left(\frac{N}{2N'}+1\right)\right\rceil N'+ & \\ \left(\frac{N}{2N'}\right)N' & \text{otherwise}\end{cases}$$
$$=\begin{cases}MN'+N' & \text{if } N=2N'\\ 2T_{12}\left(\frac{N}{2},M\right)+M\left\lceil\log\left(\frac{N}{2N'}+1\right)\right\rceil N'+ & \\ +\frac{N}{2} & \text{otherwise}\end{cases} \quad (27)$$

$$\begin{aligned} T_{12}(N,M) = & \left[M\left(\left\lceil \log\left({}^{N}/_{2N'}+1\right)\right\rceil N'\right)+{}^{N}/_{2}\right] + \\ & 2\left[M\left(\left\lceil \log\left({}^{N}/_{4N'}+1\right)\right\rceil N'\right)+{}^{N}/_{4}\right]+\ldots+ \\ & {}^{N'}/_{2}\left[M\left(\left\lceil \log\left({}^{N}/_{N'N'}+1\right)\right\rceil N'\right)+{}^{N}/_{N'}\right] \\ = & {}^{1}/_{2}M\sqrt{N}(4\sqrt{N}-\log N-4)+{}^{1}/_{4}N\log N \end{aligned} \tag{28}$$

The solution to the recurrence relation in Eq. (15) corresponding to binary search is obtained by Eq. (29).

$$\begin{aligned} T_{1}(N,M) = & \sqrt{N}\left[{}^{1}/_{2}M\sqrt{N}(\sqrt{N}-1)+{}^{1}/_{4}\sqrt{N}\log N\right] + \\ & {}^{1}/_{2}M\sqrt{N}(4\sqrt{N}-\log N-4)+{}^{1}/_{4}N\log N \\ = & {}^{1}/_{2}M\sqrt{N}(N+3\sqrt{N}-\log N-4)+{}^{1}/_{2}N\log N \end{aligned} \tag{29}$$

7.2.2 Parallel Approach

The recurrence relation of the parallel version is given by Eq. (21). The first part of the recurrence relation, *i.e.*, $T_{11}(\sqrt{N})$ is the same as Eq. (22) so the solution to Eq. (22) is obtained by Eq. (23). The second part of the recurrence relation is given by Eq. (30) which is solved using Eq. (31).

$$\begin{aligned} \underbrace{T_{\infty 2}(N)}_{N>N'} = & \begin{cases} 1+\log N'+N' & \text{if } N=2N' \\ T_{\infty 2}\left(\frac{N}{2}\right)+\left\lceil \log\left(\frac{N}{2N'}+1\right)\right\rceil (1+\log N') & \\ +\left(\frac{N}{2N'}\right)N' & \text{otherwise} \end{cases} \\ = & \begin{cases} 1+\log N'+N' & \text{if } N=2N' \\ T_{\infty 2}\left(\frac{N}{2}\right)+\left\lceil \log\left(\frac{N}{2N'}+1\right)\right\rceil & \\ (1+\log N')+\frac{N}{2} & \text{otherwise} \end{cases} \end{aligned} \tag{30}$$

$$\begin{aligned} T_{\infty 2}(N) = & \left[\left\lceil \log\left({}^{N}/_{2N'}+1\right)\right\rceil (1+\log N')+{}^{N}/_{2}\right] + \\ & \left[\left\lceil \log\left({}^{N}/_{4N'}+1\right)\right\rceil (1+\log N')+{}^{N}/_{4}\right]+\ldots+ \\ & \left[\left\lceil \log\left({}^{N}/_{N'N'}+1\right)\right\rceil (1+\log N')+{}^{N}/_{N'}\right] \\ = & {}^{1}/_{16}(16N-16\sqrt{N}+\log^{3}N+4\log^{2}N+4\log N) \end{aligned} \tag{31}$$

The solution to the recurrence relation in Eq. (21) corresponding to binary search is obtained by Eq. (32).

$$\begin{aligned} T_{\infty}(N) = & \sqrt{N}\left[{}^{1}/_{8}(8\sqrt{N}+\log^{2}N+2\log N-8)\right] + \\ & {}^{1}/_{16}(16N-16\sqrt{N}+\log^{3}N+4\log^{2}N+4\log N) \\ = & 2N-2\sqrt{N}+{}^{1}/_{8}\sqrt{N}\log^{2}N+{}^{1}/_{4}\sqrt{N}\log N+ \\ & {}^{1}/_{16}\log^{3}N+{}^{1}/_{4}\log^{2}N+{}^{1}/_{4}\log N \end{aligned} \tag{32}$$

8 Scenario – IV

We establish the recurrence relation for the serial and parallel version when N solutions are equally divided in $\sqrt{N}$ fronts such that each solution in a front is dominated by only one solution in its preceding front. As the number of fronts is more than one, so the sequential and the binary search based approaches perform differently. Let us consider $N' = \sqrt{N}$.

In this scenario, N solutions are equally divided into $\sqrt{N}$ fronts. So, in the final sorted fronts, each front has $\sqrt{N}(= 2^{\mathcal{L}/2})$ solutions, so in the merge operation from the first to the $\mathcal{L}/2^{th}$ level, all the solutions are non-dominated. So, the recurrence relation till the $\mathcal{L}/2^{th}$ level is the same as the recurrence relation for the previous scenario till the $\mathcal{L}/2^{th}$ level.

8.1 Sequential Search based Approach

Now, we obtain the recurrence relation using a sequential search based strategy for both the serial and the parallel version.

8.1.1 Serial Approach

The recurrence relation of the serial version is given by Eq. (33). This recurrence relation is in two parts. The first part corresponds to the merge operation up to the $\mathcal{L}/2^{th}$ level and the second part corresponds to the merge operation from the $\mathcal{L}/2 + 1^{th}$ to the last level.

$$T_1(N, M) = N' T_{11}(\sqrt{N}, M) + \underbrace{T_{12}(N, M)}_{N > N'} \tag{33}$$

First Part of the Recurrence Relation: Each front in the final set of fronts has $\sqrt{N}$ solutions, so the first part of the recurrence relation is the same as described by Eq. (16). The solution to Eq. (16) is obtained by Eq. (17).

Second Part of the Recurrence Relation: At the end of the merge operation at the $\mathcal{L}/2^{th}$ level, each set of fronts has a single front which has $\sqrt{N}$ solutions. After the merge operation at the $\mathcal{L}/2^{th}$ level, each set of fronts has more than one front and each front has $\sqrt{N}$ solutions. Whenever a merge operation is performed after the $\mathcal{L}/2^{th}$ level, only the solutions of the first front in $\mathcal{F}'$ are compared with the solutions of $\mathcal{F}$. The solutions of the remaining fronts in $\mathcal{F}'$ are added directly to $\mathcal{F}$ because of the dominance relationship as discussed in [17]. The solutions of a front $F' \in \mathcal{F}'$ are compared with respect to all the solutions of each front in $\mathcal{F}$ and dominated by the last solution of each front because each solution in a front is dominated by only one solution in its preceding front.

In a merge operation where both sets of fronts have $N/2$ solutions, only $\sqrt{N}$ solutions in the first front in $\mathcal{F}'$ are compared with respect to all the solutions in all $N/2\sqrt{N}$ fronts in $\mathcal{F}$. So, $T_{\text{dom}}(N, M) = M\,(N/2\sqrt{N})\,\sqrt{N}\sqrt{N} = M\,(N/2)\,\sqrt{N}$.

All the solutions from $\mathcal{F}'$ need to be inserted into $\mathcal{F}$ so, $T_{\text{insert}}(N) = {}^{N}/_{2}$. Thus, the recurrence relation corresponding to the second part is given by Eq. (34) which is solved using Eq. (35).

$$\begin{aligned}\underbrace{T_{12}(N,M)}_{N>N'} &= \begin{cases} MN'N'+N' & \text{if } N{=}2N' \\ 2T_{12}\left(\frac{N}{2},M\right)+M\left(\frac{N}{2N'}\right)N'N'+\left(\frac{N}{2N'}\right)N' & \text{otherwise}\end{cases} \\ &= \begin{cases} MN'N'+N' & \text{if } N{=}2N' \\ 2T_{12}\left(\frac{N}{2},M\right)+M\left(\frac{N}{2}\right)N'+{}^{N}/_{2} & \text{otherwise}\end{cases}\end{aligned} \tag{34}$$

$$\begin{aligned}T_{12}(N,M) &= \left[M\left({}^{N}/_{2}\right)N'+{}^{N}/_{2}\right]+2\left[M\left({}^{N}/_{4}\right)N'+{}^{N}/_{4}\right]+\ldots \\ &\quad +{}^{N'}/_{2}\left[M\left({}^{N}/_{N'}\right)N'+{}^{N}/_{N'}\right] \\ &= {}^{1}/_{4}MN\sqrt{N}\log N+{}^{1}/_{4}N\log N\end{aligned} \tag{35}$$

The solution to the recurrence relation in Eq. (33) is obtained by Eq. (36).

$$\begin{aligned}T_1(N,M) &= \sqrt{N}\left[{}^{1}/_{2}M\sqrt{N}(\sqrt{N}-1)+{}^{1}/_{4}\sqrt{N}\log N\right]+ \\ &\quad {}^{1}/_{4}MN\sqrt{N}\log N+{}^{1}/_{4}N\log N \\ &= {}^{1}/_{4}MN(\sqrt{N}\log N+2\sqrt{N}-2)+{}^{1}/_{2}N\log N\end{aligned} \tag{36}$$

The time complexity of the serial version (sequential search based) in this scenario is $\mathcal{O}(MN\sqrt{N}\log N)$ (see Eq. (36)) whereas the time complexity in the previous scenario is $\mathcal{O}(MN\sqrt{N})$ (see Eq. (20)). This is because, in the current scenario, each solution in a front is dominated by only one solution in its previous front. So, a solution is compared with respect to all the solutions in its previous fronts. However, in the previous scenario, each solution in a front is dominated by all the solutions in its previous front. So, a solution is compared with only one solution in its previous fronts.

8.1.2 Parallel Approach

The time complexity of the parallel version in this scenario is the same as that of the parallel version in the previous scenario which is given by Eq. (26).

Regardless of the difference in these two scenarios, the time complexity of the parallel version is the same. The merge operation till the ${}^{\mathcal{L}}/_{2^{th}}$ level is the same in both scenarios. After the merge operation at the ${}^{\mathcal{L}}/_{2^{th}}$ level, in the parallel version in both scenarios, a solution sol' of the first front in $\mathcal{F}'$ is simultaneously compared with respect to all the solutions of a front $F \in \mathcal{F}$. The dominance relation of sol' with respect to the solutions of F is stored in an array and then, this array is processed in a parallel manner. So, the time complexity in the second phase in both scenarios is also the same.

8.2 Binary Search based Approach

Here, the recurrence relation is established using a binary search based strategy. In the binary search based approach, the merge operations up to the $\mathcal{L}/2^{th}$ level remain the same as in the sequential search based approach because up to the $\mathcal{L}/2^{th}$ level all the solutions are in a single front. After the $\mathcal{L}/2^{th}$ level, in a merge operation where both sets of fronts have $N/2$ solutions, only $\sqrt{N}$ solutions in the first front in $\mathcal{F}'$ are compared with respect to all the solutions in $\lceil \log(N/2\sqrt{N}+1) \rceil$ fronts in $\mathcal{F}$.

8.2.1 Serial Approach

The recurrence relation of the serial version is given by Eq. (33). The first part of the recurrence relation, *i.e.*, $T_{11}(\sqrt{N}, M)$ is the same as Eq. (16) so the solution to Eq. (16) will be the same as Eq. (17). The second part of the recurrence relation is given by Eq. (37) which is solved using Eq. (38).

$$\begin{aligned}\underbrace{T_{12}(N,M)}_{N>N'} &= \begin{cases} MN'N'+N' & \text{if } N=2N' \\ 2T_{12}\left(\frac{N}{2},M\right)+M\left\lceil \log\left(\frac{N}{2N'}+1\right)\right\rceil N'N'+ & \\ \left(\frac{N}{2N'}\right)N' & \text{otherwise} \end{cases} \\ &= \begin{cases} MN'N'+N' & \text{if } N=2N' \\ 2T_{12}\left(\frac{N}{2},M\right)+M\left\lceil \log\left(\frac{N}{2N'}+1\right)\right\rceil N'N' & \\ +\frac{N}{2} & \text{otherwise} \end{cases} \end{aligned} \tag{37}$$

$$\begin{aligned} T_{12}(N,M) = &\left[M\left\lceil \log\left(N/2N'+1\right)\right\rceil N'N'+N/2\right]+ \\ &2\left[M\left\lceil \log\left(N/4N'+1\right)\right\rceil N'N'+N/4\right]+\ldots \\ &+N'/2\left[M\left\lceil \log\left(N/N'N'+1\right)\right\rceil N'N'+N/N'\right] \\ =&1/2MN(4\sqrt{N}-\log N-4)+1/4N\log N \end{aligned} \tag{38}$$

The solution to the recurrence relation in Eq. (33) corresponding to binary search is obtained in Eq. (39).

$$\begin{aligned} T_1(N,M) =& \sqrt{N}\left[1/2M\sqrt{N}(\sqrt{N}-1)+1/4\sqrt{N}\log N\right]+ \\ & 1/2MN(4\sqrt{N}-\log N-4)+1/4N\log N \\ =&1/2MN(5\sqrt{N}-\log N-5)+1/2N\log N \end{aligned} \tag{39}$$

8.2.2 Parallel Approach

The time complexity of the parallel version in this scenario is the same as the time complexity of the parallel version in the previous scenario which is given by Eq. (32). The merge operation till the $\mathcal{L}/2^{th}$ level is the same in both

scenarios. After the $\mathcal{L}/2^{th}$ level, in the parallel version in both scenarios, a solution sol' of a front $F' \in \mathcal{F}'$ is simultaneously compared with respect to all the solutions of the first front in $\mathcal{F}$. So, the time complexity in the second phase in both scenarios is the same.

9 Space Complexity

In this section, we discuss the space complexity of the parallel version of the approach. Merge sort is used in the first phase so the space complexity of the first phase is $\mathcal{O}(N)$. To obtain the dominance matrix, a solution is compared with another solution using a Boolean array of size M. The space required to store this array is $\mathcal{O}(M)$. There are N^2 pairs of solutions, so the space required to obtain the dominance matrix is $\mathcal{O}(MN^2)$. The size of the dominance matrix is $N \times N$ so the space required to store the dominance matrix is $\mathcal{O}(N^2)$.

At each level, different merge operations took place simultaneously. A solution of front $F' \in \mathcal{F}'$ is simultaneously compared with respect to all the solutions of a front $F \in \mathcal{F}$ and the dominance relation of this particular solution with respect to all the solutions of F is stored in an array. The maximum number of solutions in a front in both sets of fronts $\mathcal{F}$ and $\mathcal{F}'$ is $N/2$. So, when a solution of front $F' \in \mathcal{F}'$ is simultaneously compared with respect to all the solutions of a front $F \in \mathcal{F}$, then the size of the array which stores the dominance relationship of a solution of F' with respect to all the solutions of F is $N/2$. Each solution of F' requires an array of size $N/2$ to store the dominance relationship and there are maximum $N/2$ solutions in a front $F' \in \mathcal{F}'$, so the space required for this array is $N/2 \times N/2$ which is $\mathcal{O}(N^2)$. Thus, the overall space complexity of the parallel version of the approach is $\mathcal{O}(MN^2)$.

10 Comparison With Some Other Approaches

Various approaches for non-dominated sorting have the parallelism property as explained in Section 1. We obtain the worst case time complexity of the parallel version of some of the approaches considering the PRAM-CREW model so that a comparison can be made.

CNS (Constraint Non-dominated Sorting) [23] is used to assign a rank which is called the constrained non-dominated rank (CNR). In general, the rank which we obtained after applying various non-dominated sorting approaches [2, 4, 6, 7, 9, 11, 14, 26, 27, 29, 30, 32, 33] is called Pareto rank [23]. CNR considers Pareto rank and constraint rank to obtain its value. So, the time complexity of CNS depends on the time complexity of the algorithm which is used to obtain the Pareto rank. If we use parallel approach to find the CNR of the solutions in a population, then also the time complexity of the parallel version depends on the time complexity of the parallel version of the Pareto ranking approach. So, the time complexity of the parallel version of CNS cannot be better than the time complexity of the Pareto ranking approach which is used to obtain the Pareto rank of the solutions. Thus, until or

unless we propose a new efficient non-dominated sorting approach to obtain the Pareto rank of the solutions, CNS cannot perform better than the existing non-dominated sorting approaches.

T-ENS (Tree based Efficient Non-dominated Sorting Approach) [33] works in two phases. When parallelism is considered in T-ENS, the time complexity of the parallel version of the first phase is $\mathcal{O}(M \log^3 N)$ using parallel merge sort where the merge operation is itself implemented in a parallel manner. The maximum number of processors required in the first phase is N. In case of T-ENS, when solution sol_j is compared with sol_i, then we find the minimal index $m(1 \leq m \leq M)$ of the objective for which the value of sol_j is smaller than sol_i. For example, let's assume that we have two solutions $sol_i = \{1, 3, 4, 2\}$ and $sol_j = \{6, 4, 2, 1\}$. In this case, the minimum index m for which sol_j has a smaller value than sol_i is 3. This index is obtained so that the solutions in a front can be represented as a tree. This index between two solutions can be obtained in $\mathcal{O}(M)$ time by comparing each objective sequentially. This type of index value between each pair of solutions can be obtained in parallel. Similar to the *dominance matrix* which stores the dominance relationship between all pairs of solutions, we can have a matrix which stores the minimum index value when all pairs of solutions are compared. Let us call this matrix *Index Matrix*. The *index matrix* can be obtained in $\mathcal{O}(M)$ time in parallel. From the *index matrix*, the minimum index value, when comparing two solutions can be obtained in $\mathcal{O}(1)$ time. As there are N^2 pairs of solutions, so the maximum number of processors required to obtain *index matrix* is N^2. In the worst case of T-ENS, all the solutions are in different fronts and the i^{th} solution is compared with the solutions of all the previous fronts sequentially. Thus, the time complexity of the parallel version in the worst case is given by Eq. (40).

$$\begin{aligned} T_{\text{T-ENS\_rank\_parallel}} &= 1 + \sum\nolimits_{i=2}^{N} [(i-1)+1] \\ &= 1 + \frac{1}{2}N(N-1) + (N-1) \\ &= \mathcal{O}(N^2) \end{aligned} \tag{40}$$

In the worst case of T-ENS, each front has a single solution so, the parallel version also behaves as its sequential counterpart except for obtaining the minimum index value while comparing two solutions in $\mathcal{O}(1)$ time considering index matrix. So, only a single processor is required. Thus, the overall worst case time complexity of the parallel version is $\mathcal{O}(M \log^3 N) + \mathcal{O}(M) + \mathcal{O}(N^2) = \mathcal{O}(N^2)$. The maximum number of processors required by T-ENS is N^2.

ENS-NDT (Efficient Non-dominated Sort with Non-Dominated Tree) [9] works in three phases. When parallelism is considered in ENS-NDT, the time complexity of the parallel version of the first phase is $\mathcal{O}(M \log^3 N)$ using parallel merge sort where the merge operations are itself implemented in a parallel manner. The maximum number of processors required to sort the solutions based on the last objective in a parallel manner is N. The method to create the prebalanced split is recursive in nature so parallelism is possible in this method, too. The time complexity of the parallel version of the prebalanced

split is obtained by Eq. (41). The maximum number of processors required for the prebalance is $\frac{1}{2}N$.

$$\begin{aligned} T_{\text{splits\_parallel}} &= N\log N + \left(\frac{N}{2}\log\frac{N}{2}\right) + \left(\frac{N}{4}\log\frac{N}{4}\right) + \ldots + \left(\frac{N}{N/2}\log\frac{N}{N/2}\right) \\ &= \left[N\log N + \frac{N}{2}\log N + \frac{N}{4}\log N + \ldots + \frac{N}{N/2}\log N\right] - \\ &\quad \left[\frac{N}{2}\log 2 + \frac{N}{4}\log 4 + \ldots + \frac{N}{N/2}\log\frac{N}{2}\right] \\ &= 2(N\log N - N + 1) \\ &= \mathcal{O}(N\log N) \end{aligned} \tag{41}$$

In the last phase, the solutions are assigned to their respective front. In the worst case, the number of fronts is one. In the worst case, the first $M-2$ objective values of all the solutions are the same and the last two objective values are such that they fulfill the non-dominated criteria [9]. To achieve parallelism in the third phase, the solution which needs to be assigned to a front, can be compared with respect to all the solutions of a particular front simultaneously and then it can be decided whether that solution can be inserted into the particular front or not. In the case of the parallel version of ENS-NDT, the *dominance matrix* is obtained so that the dominance relationship between two solutions can be obtained in $\mathcal{O}(1)$ time. The time complexity of obtaining the *dominance matrix* in a parallel manner is $\mathcal{O}(\log M)$. The maximum number of processors required to obtain the *dominance matrix* in a parallel manner is $2MN^2$.

In the worst case, the first solution is assigned to the first front without comparing with respect to any other solution. The i^{th} solution which needs to be ranked, is compared with all the $i-1$ solutions of the first front simultaneously. This can be performed in $\mathcal{O}(1)$ time in a parallel manner as the dominance relation between two solutions can be obtained in $\mathcal{O}(1)$ time using the *dominance matrix*. After comparing the i^{th} solution with respect to all the $i-1$ solutions simultaneously, the dominance nature of the i^{th} solution with all the solutions is obtained. As the i^{th} solution is compared with respect to $i-1$ solutions, so the dominance nature can be obtained in a parallel manner in $\mathcal{O}(\log i)$ time. After deciding that the i^{th} solution is non-dominated with respect to all the $i-1$ solutions, it is added to the front in $\mathcal{O}(1)$ time. The time complexity of the parallel version in the third phase is given by Eq. (42). The maximum number of processors required to assign rank to all the solutions in the third phase is $N-1$.

$$\begin{aligned} T_{\text{ENS-NDT\_rank\_parallel}} &= 1 + \sum\nolimits_{i=2}^{N} \left[1 + \left\lceil \log i \right\rceil + 1\right] \\ &= 1 + (N-1) + N\log N - (N-1) + (N-1) \\ &= N\log N + N = \mathcal{O}(N\log N) \end{aligned} \tag{42}$$

Thus, the overall time complexity of the parallel version of ENS-NDT is $\mathcal{O}(M \log^3 N)+\mathcal{O}(N \log N)+\mathcal{O}(\log M)+\mathcal{O}(N \log N) = \mathcal{O}(N \log N)$. The maximum number of processors required by ENS-NDT is $2MN^2$.

The worst case time complexity of the parallel version of ENS is $\mathcal{O}(N^2)$ [16]. From this analysis it is clear that the worst time complexity of the parallel version of DCNS is better than the worst case time complexity of the parallel version of some of the other approaches.

11 Conclusions & Future Work

In this paper, we have thoroughly explored and analyzed the parallelism in the DCNS approach considering the PRAM CREW model. The parallelism is discussed in both phases of the DCNS approach. The best case time complexity of the serial version of the DCNS approach (based on binary search) is $\mathcal{O}(N \log N + MN)$. This happens when all the solutions are in different fronts. Here, the values of the first objective for each of the solutions are different. This best case time complexity is better than the best case time complexity of the previous approaches for $M > 3$. The worst case time complexity of the serial version is $\mathcal{O}(MN^2)$ when all the solutions are non-dominated.

The time complexity of the parallel version of the second phase is $\mathcal{O}(N)$. The time required to compute the dominance matrix is $\mathcal{O}(\log M)$ and the time complexity of the first phase is $\mathcal{O}(\log^3 N)$. Thus, the overall time complexity of the parallel version of the approach is $\mathcal{O}(\log^3 N) + \mathcal{O}(\log M) + \mathcal{O}(N) = \mathcal{O}(\log M + N)$. The space complexity of the parallel version of our approach is $\mathcal{O}(MN^2)$. The time complexity of the parallel version of non-dominated sorting is proved to be $\mathcal{O}(M+N)$ in [28]. We have analyzed the time complexity of the serial and parallel version in four different scenarios. In the last two scenarios, the time complexity of the serial version differs; however, the time complexity of the parallel version remains the same. The maximum number of processors required for the parallel version of the DCNS approach is $2MN^2$.

In this paper, we have performed a theoretical analysis, so as part of our future work, we would like to implement the parallel version of the approach and to measure the actual speedup. It would also be interesting to use the divide-and-conquer based approach in BOS and other approaches for non-dominated sorting to achieve the parallelism if possible. The incremental non-dominated sorting problem can also be a potential area of future work. It would be really interesting to compare different non-dominated sorting approaches based on their parallel version. In this way, we can analyze which algorithm is better in a parallel environment as compared to others.

Acknowledgements

The second author gratefully acknowledges support from CONACyT grant no. 2016-01-1920 (*Investigación en Fronteras de la Ciencia 2016*).

References

1. Bao C, Xu L, Goodman ED, Cao L (2017) A Novel Non-Dominated Sorting Algorithm for Evolutionary Multi-Objective Optimization. Journal of Computational Science 23:31–43
2. Buzdalov M, Shalyto A (2014) A Provably Asymptotically Fast Version of the Generalized Jensen Algorithm for Non-dominated Sorting. In: Parallel Problem Solving from Nature - PPSN XIII, 13th International Conference, Springer. Lecture Notes in Computer Science Vol. 8672, Ljubljana, Slovenia, pp 528–537
3. Cormen TH, Leiserson CE, Rivest RL, Stein C (2009) Introduction to algorithms. MIT press, Cambridge, Massachusetts, USA, ISBN: 978-0-262-03384-8
4. Deb K, Pratap A, Agarwal S, Meyarivan T (2002) A Fast and Elitist Multiobjective Genetic Algorithm: NSGAII. IEEE Transactions on Evolutionary Computation 6(2):182–197
5. Drozdik M, Akimoto Y, Aguirre H, Tanaka K (2015) Computational Cost Reduction of Nondominated Sorting Using the M-Front. IEEE Transactions on Evolutionary Computation 19(5):659–678
6. Fang H, Wang Q, Tu YC, Horstemeyer MF (2008) An Efficient Non-dominated Sorting Method for Evolutionary Algorithms. Evolutionary Computation 16(3):355–384
7. Fortin FA, Greiner S, Parizeau M (2013) Generalizing the Improved Run-Time Complexity Algorithm for Non-Dominated Sorting. In: 2013 Genetic and Evolutionary Computation Conference (GECCO'2013), ACM Press, New York, USA, pp 615–622, ISBN: 978-1-4503-1963-8
8. Gupta S, Tan G (2015) A Scalable Parallel Implementation of Evolutionary Algorithms for Multi-Objective Optimization on GPUs. In: 2015 IEEE Congress on Evolutionary Computation (CEC'2015), IEEE Press, Sendai, Japan, pp 1567–1574, ISBN: 978-1-4799-7492-4
9. Gustavsson P, Syberfeldt A (2018) A New Algorithm Using the Non-Dominated Tree to Improve Non-Dominated Sorting. Evolutionary Computation 26(1):89–116
10. JáJá J (1992) An Introduction to Parallel Algorithms, vol 17. Addison-Wesley Reading
11. Jensen MT (2003) Reducing the Run-Time Complexity of Multiobjective EAs: The NSGA-II and Other Algorithms. IEEE Transactions on Evolutionary Computation 7(5):503–515
12. Kumar V, Grama A, Gupta A, Karypis G (1994) Introduction to Parallel Computing: Design and Analysis of Algorithms, vol 400. Benjamin/Cummings Redwood City
13. Kung HT, Luccio F, Preparata FP (1975) On Finding the Maxima of a Set of Vectors. Journal of the ACM (JACM) 22(4):469–476
14. McClymont K, Keedwell E (2012) Deductive Sort and Climbing Sort: New Methods for Non-Dominated Sorting. Evolutionary Computation 20(1):1–26

15. Mikloško J, Kotov VE (1984) Algorithms, Software and Hardware of Parallel Computers
16. Mishra S, Coello Coello CA (2018) P-ENS: Parallelism in Efficient Non-Dominated Sorting. In: 2018 IEEE Congress on Evolutionary Computation (CEC'2018), IEEE Press, Rio de Janeiro, Brazil, pp 508–515, ISBN: 978-1-5090-6017-7
17. Mishra S, Saha S, Mondal S (2016) Divide and Conquer Based Non-Dominated Sorting for Parallel Environment. In: 2016 IEEE Congress on Evolutionary Computation (CEC'2016), IEEE Press, Vancouver, Canada, pp 4297–4304, ISBN: 978-1-5090-0623-6
18. Mishra S, Mondal S, Saha S, Coello CAC (2018) GBOS: Generalized Best Order Sort Algorithm for Non-dominated Sorting. Swarm and Evolutionary Computation, doi.org/10.1016/j.swevo.2018.06.003
19. Mishra S, Saha S, Mondal S (2018) MBOS: Modified Best Order Sort Algorithm for Performing Non-dominated Sorting. In: 2018 IEEE Congress on Evolutionary Computation (CEC'2018), IEEE Press, Rio de Janeiro, Brazil, pp 725–732, ISBN: 978-1-5090-6017-7
20. Mishra S, Saha S, Mondal S, Coello CAC (2018) A Divide-and-Conquer Based Efficient Non-Dominated Sorting Approach. Swarm and Evolutionary Computation, doi.org/10.1016/j.swevo.2018.08.011
21. Moreno J, Ortega G, Filatovas E, Martínez J, Garzón E (2018) Improving the Performance and Energy of Non-Dominated Sorting for Evolutionary Multiobjective Optimization on GPU/CPU Platforms. Journal of Global Optimization pp 1–19
22. Niculescu V (2007) Data-distributions in powerlist theory. In: International Colloquium on Theoretical Aspects of Computing, Springer, pp 396–409
23. Ning W, Guo B, Yan Y, Wu X, Wu J, Zhao D (2017) Constrained Multi-Objective Optimization Using Constrained Non-dominated Sorting Combined with an Improved Hybrid Multi-Objective Evolutionary Algorithm. Engineering Optimization 49(10):1645–1664
24. Ortega G, Filatovas E, Garzon EM, Casado LG (2017) Non-Dominated Sorting Procedure for Pareto Dominance Ranking on Multicore CPU and/or GPU. Journal of Global Optimization 69(3):607–627
25. Palakonda V, Pamulapati T, Mallipeddi R, Biswas PP, Veluvolu KC (2017) Nondominated Sorting Based on Sum of Objectives. In: 2017 IEEE Symposium Series on Computational Intelligence (SSCI), IEEE, pp 1–8
26. Roy PC, Islam MM, Deb K (2016) 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, ACM Press, Denver, Colorado, USA, pp 1113–1120, ISBN: 978-1-4503-4323-7
27. Roy PC, Deb K, Islam MM (2018) An Efficient Nondominated Sorting Algorithm for Large Number of Fronts. IEEE Transactions on Cybernetics doi: 10.1109/TCYB.2017.2789158

28. Smutnicki C, Rudy J, Zelazny D (2014) Very Fast Non-dominated Sorting. Decision Making in Manufacturing and Services 8(1-2):13–23
29. Srinivas N, Deb K (1994) Multiobjective Optimization Using Nondominated Sorting in Genetic Algorithms. Evolutionary Computation 2(3):221–248
30. Tang S, Cai Z, Zheng J (2008) A Fast Method of Constructing the Non-dominated Set: Arena's Principle. In: 2008 Fourth International Conference on Natural Computation, IEEE Computer Society Press, Jinan, China, pp 391–395, ISBN: 978-0-7695-3304-9
31. Van Veldhuizen DA, Zydallis JB, Lamont GB (2003) Considerations in Engineering Parallel Multiobjective Evolutionary Algorithms. IEEE Transactions on Evolutionary Computation 7(2):144–173
32. Zhang X, Tian Y, Cheng R, Yaochu J (2015) An Efficient Approach to Nondominated Sorting for Evolutionary Multiobjective Optimization. IEEE Transactions on Evolutionary Computation 19(2):201–213
33. Zhang X, Tian Y, Cheng R, Jin Y (2018) A Decision Variable Clustering-Based Evolutionary Algorithm for Large-Scale Many-Objective Optimization. IEEE Transactions on Evolutionary Computation 22(1):97–112
34. Zhou Y, Chen Z, Zhang J (2017) Ranking Vectors by Means of the Dominance Degree Matrix. IEEE Transactions on Evolutionary Computation 21(1):34–51