Self-organized Invasive Parallel Optimization

Sanaz Mostaghim
Institute AIFB
Karlsruhe Institute of Technology
Karlsruhe, Germany
sanaz.mostaghim@kit.edu

Friederike Pfeiffer
Institute AIFB
Karlsruhe Institute of Technology
Karlsruhe, Germany
friederike.pfeiffer@kit.edu

Hartmut Schmeck
Institute AIFB
Karlsruhe Institute of Technology
Karlsruhe, Germany
hartmut.schmeck@kit.edu

ABSTRACT

Self-organized Invasive Parallel Optimization (SIPO) is a new framework for solving optimization problems on parallel platforms. In contrast to existing approaches, the resources in SIPO are self-organized and represented as a unified resource to the user who specifies the optimization problem and its preferences to the system. SIPO starts working with one resource and automatically divides the optimization task stepwise into smaller tasks which are assigned to more resources. This job assignment is decided on demand by the resources. The novelty here is that there is no need to specify the number of parallel computing resources in the beginning of the optimization. This number is estimated during the optimization process by the resources. The proposed new framework of SIPO is described in this paper with respect to multi-objective optimization problems but it has a much larger scope. A comparative evaluation of using SIPO in multi-objective optimization problems shows that this adaptive approach can obtain equally good or sometimes even better solutions than other parallel and non-parallel methods which are not self-organized.

Categories and Subject Descriptors

1.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods and Search

General Terms

Algorithms, Performance

Keywords

self-organized optimization

1. INTRODUCTION

In recent years, we have been witnessing highly parallel systems for computations such as grids, clouds and multi/many-core systems [2]. These systems facilitate parallel computation of very difficult and time consuming scientific problems (e.g., optimizing parameters in pharmaceutical products) which could not be solved in the past. In the future, even more computing resources will be available to solve even harder problems. However, as it is difficult for users to explicitly control highly parallel systems, we need to design new algorithms and frameworks to let the computing resources work in a self-organized way. Optimization and particularly multi-objective optimization algorithms as major tools in computational science can significantly benefit from performing computations on such parallel platforms.

In this paper, we introduce a new framework for parallel optimization in a scenario where the computing resources cooperate to solve an optimization problem. The computation starts with one available computing resource to which the problem and the user preferences are given. According to the user preferences, the starting processor performs a rough optimization and divides its partition into smaller partitions. It assigns the partitions for further optimization to other processors in the platform. As soon as a processing unit gets a job for optimization, it performs the same procedure described above and either assigns new jobs to other processors or stops the optimization process in its partition. In SIPO, the approximation of the optimal solutions gets more precise over the successive divisions of the tasks. From another point of view, the optimization starts from one computing resource and then successively occupies more resources for computation until the computing resources can not obtain any better solutions. The parallel resources are released one by one until there is no computing resource required for the optimization and in fact, the system stops automatically. In addition, the resources partition the tasks and synchronize themselves over the optimization process. We call this approach **Self-organized Invasive Parallel Optimization (SIPO)** as it is inspired by invasive computing in many-core systems [1]. In contrast to other approaches in parallel optimization where users set a preliminary number of resources which are available in a laboratory, cluster or grid and run the parallel algorithm on them, our approach has the advantage that the number of required processors need not to be known in advance. For instance, in some versions of the island model, the user divides the parameter or objective space into a certain number of sub-spaces which are then assigned to processors [19, 5, 13]. For unknown problems, dividing the space is very difficult, yet very important for estimating the number of required resources to obtain an acceptable set of solutions. In our approach, these tasks are performed by the resources on demand.

BADS'11, June 14, 2011, Karlsruhe, Germany.

SIPO is a general framework for self-organized parallel optimization. However, in order to validate the functionality of SIPO, we investigate multi-objective problems in this paper. The main property of the optimization method in SIPO is that it must be able to focus inside a given interval. This kind of optimization for multi-objective problems is studied in the context of guided multi-objective evolutionary algorithms [4] and preference-based optimization e.g., [16, 22, 20] where the user specifies a desired area. However, focusing and keeping the solutions in an interval in the objective space is very difficult as the optimization performs the search in the parameter space. For this purpose, Particle Swarm Optimization methods [16] are shown to be suitable and therefore we investigate a variant of PSO in our paper.

In order to evaluate SIPO, we select a platform of parallel resources which communicate through a shared memory. We examine SIPO on three different kinds of continuous multi-objective optimization problems with different shapes of Pareto-fronts and observe the behavior of SIPO on several parameters. In the experiments, we measure the maximum number of required parallel processors on our self-organized platform. Additionally, the speed-up factor is measured by comparing our results with the results of baseline parallel and non-parallel algorithms.

This paper is organized as follows: After a short description of multi-objective problems below, we outline our approach Self-organized Invasive Parallel Optimization (SIPO) in Section 2. The division process in SIPO is explained more into details in Section 3 and the optimization algorithm employed in SIPO is explained in Section 4. Section 5 contains the experiments and the analysis of the results. The paper is concluded in the last section.

Multi-objective Problems

Typically a Multi-Objective Problem (MOP) involves several objectives which have to be optimized simultaneously, i.e. the objective function is multi-dimensional $\vec{f} : \mathbb{R}^n \to \mathbb{R}^m$:

$$\min_{\vec{x} \in S \subset \mathbb{R}^n} \qquad f_i(\vec{x}) \quad \text{for } i = 1 \dots m$$

We denote the image of S by $Z \subset \mathbb{R}^m$ and call it the objective space, the elements of Z are called objective vectors. Since we are dealing with MOPs, there is not generally one global optimum but a set of so-called **Pareto optimal solutions**. A decision vector $\vec{x}_1 \in S$ is called **Pareto-optimal** if there is no other decision vector $\vec{x}_2 \in S$ that **dominates** it: $\vec{x}_1$ is said to dominate $\vec{x}_2$ if $\vec{x}_1$ is not worse than $\vec{x}_2$ in all of the objectives and it is strictly better than $\vec{x}_2$ in at least one objective. An objective vector is called Pareto-optimal if the corresponding decision vector is Pareto-optimal.

2. SELF-ORGANIZED INVASIVE PARALLEL OPTIMIZATION (SIPO)

The most important elements in designing a parallel optimization algorithm are: 1- task partitioning, 2- task scheduling and 3- task synchronization [6]. In this paper we aim to design SIPO so that the computing resources collaborate and communicate with each other and perform the above tasks themselves without involvement of the users. This is in contrast to most of the existing algorithms where these tasks are pre-determined by users [19, 14].

To start SIPO, the user is asked to give a rough estimation of the position of the Pareto-front in terms of one objective in the objective space. This can be an interval, for instance $[0, 10]$ for one of the objectives with the true Pareto-front in $[0, 1]$. This rough estimation is only used in the beginning of the algorithm and can even contain an infeasible area. In SIPO, every processing unit to which an interval (job) is assigned must perform the following four steps:

1. Optimize inside a given interval to find a rough estimation of the optimal solutions
2. Communicate the results to other resources through a shared memory
3. Evaluate the obtained results in the interval: either select the interval for further divisions or go idle
4. Assign jobs to other resources: in case of further divisions, divide the interval into a number of smaller intervals, take one interval for itself and assign the rest of the intervals to other computing resources

In the above steps, the resources require two kinds of communication: A) communicating the results and B) assigning jobs to other resources. Communication of the results between the processors is achieved through a global archive in a shared memory. The global archive is accessible to all the computing resources and can be updated by them. As soon as the optimization process in the given interval is finished, the obtained results are written in the global archive. Thereby, the global archive is kept dominated-free. By using the concept of shared memory, it is not necessary to synchronize the resources and SIPO can be performed on heterogeneous as well as on homogeneous resources. The other communication type in SIPO concerns the job assignment strategy, i.e., a processing unit should be able to find idle processors and assign jobs (intervals) to them. This can be performed in different ways. For instance, every processor that is in idle mode can be reached by other processors through a network addressing methodology such as anycast or unicast. Another communication mechanism is to store a list of idle processors in the shared memory. Before a processor goes to idle mode, it stores its address in this list which can be accessed by a processor looking for idle resources. The problem-heap and tuple space approaches [12, 11] are other and well-known possibilities for self-organized job assignment which is to write the optimization intervals (jobs) in a shared memory called problem-heap or tuple space which can be accessed, read and executed by other processors.

SIPO is a mixture of island and master-slave model of parallel optimization [19, 21]: every computing resource runs an optimization algorithm on a subproblem, hence the island model. In addition, every computing resource can assign jobs to other resources as in a master-slave model.

Each computing resource to which a job is assigned for optimization runs the same algorithm as illustrated in Algorithm 1. In fact, the division process continues until no interval survives the selection mechanism, i.e., there is no interval to be optimized. As a consequence, the algorithm stops automatically.

The output of Algorithm 1 is the global archive (A) located in the shared memory. Function *Optimize*(I) indicates an optimization algorithm which focuses on a given interval I and produces a local archive (a) containing non-dominated solutions. The solutions of this optimization algorithm must be only a rough approximation of the optimal

Algorithm 1: SIPO

Input: I given Interval
Output: A global archive

$a := Optimize(I)$
Decide if I is kept for further divisions:
if *(SelectInterval*(I, A, a)*== TRUE)* **then**
  $A := Update(A, a)$
  $List_I = Divide(I, NumofDivisions)$
  $Assign\text{-}Jobs(List_I, NumofDivisons)$
else
  $A := Update(A, a)$
  go idle
end

solutions in the interval. By the Function *Update*, the output of the optimization a is merged into a dominated-free global archive A. Before updating the global archive A, the Function *SelectInterval*(I, A, a) decides if the interval I is kept for further divisions or not (this is explained more into detail in the next section). If the result is true, the Function *Divide*$(I, NumofDivisons)$ divides the input interval I into $NumofDivisions$ intervals listed in a *List*. The Function *Assign-Jobs* indicates the job assignment strategy explained above.

SIPO is a decentralized approach, but from a central point of view, this invasive optimization is like successive divisions of intervals through several iterations (depth) where each interval is assigned to a processing unit.

Example: Suppose, we start from one computing resource containing one interval I_0 and divide the selected intervals to 2 intervals each time the intervals are divided (Figure 1). The parameter d indicates the depth (age) of divisions. The intervals selected for further divisions are marked in gray colors. The maximum value for depth parameter d is 5 and the maximum number of parallel intervals (required processors) is indicated by the maximum number of intervals as 8 at depth 3. Altogether, we require 8 parallel processors which are run simultaneously. Overall, an optimization algorithm is run for 20 times. □

3. SPACE DIVISION

Space division is the most straightforward way of assigning partitions to different processors. The first known approach in multi-objective optimization is proposed by Branke et al. [5] where the objective space is divided into cones. Other similar approaches are reported, e.g. in [19, 10].

In our approach, the objective space is divided in terms of one objective[1]. For starting our division process, the user has to give a rough estimation about the position of the Pareto-front. In addition, the user can specify the number of processing units he wants to start with. In the following, we explain how the division and deletion of the intervals happen in our approach.

Interval Selection

The Function *SelectInterval*(I, A, a) decides, as explained in Algorithm 1, whether an interval can be divided further into

[1]This can be employed to all of the objectives resulting in many small intervals.

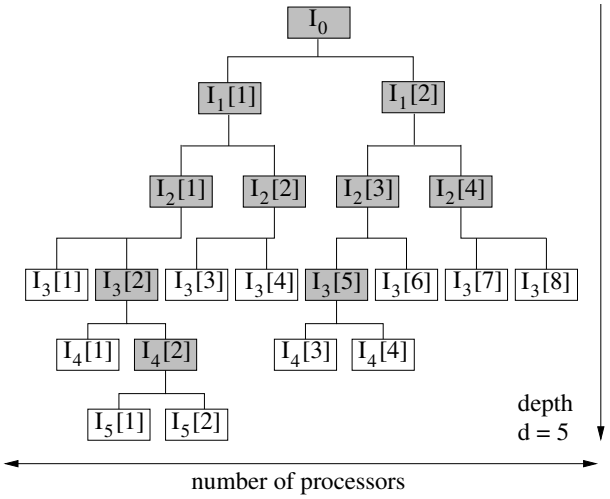


Figure 1: Example: Each box indicates an interval assigned to a processor for optimization. Through a selection process the gray colored boxes are further divided into two smaller intervals.

smaller intervals or not. For making this decision, the results a obtained in the interval are analyzed and compared with the stored results in the global archive A. The interval I is kept for further divisions, if:

1. a part of the global archive A is located within the interval (except for depth zero) and
2. some improvements of the results a in terms of quality are observed by comparing them with the results in the global archive A.

For case (2), we employ the hypervolume measurement (smetric in [23]). We compute the hypervolume of the results in a and compare them with the hypervolume of the results of A which are also located in the interval. If the difference is greater than a threshold value T, the interval survives the selection mechanism for further divisions. The value of T can be set as a constant or can adaptively change based on the depth of the interval. In the beginning, at depth 1 or 2, the solutions in a have better quality than those in A (global archive), therefore we want to keep the interval and further divide it. At this stage, the threshold value must be very small so that almost all the intervals can survive. For higher depth values such as 7 (an interval is divided 7 times), it is desired to stop if the results are not significantly better. Therefore, T must be large enough. Thereby, T can adaptively change over depth with $T = sL \times d^2$ with *significantLevel* $sL = 0.005$.

One could select other methods for computing the improvement inside an interval, for instance comparing the generational distance [7] between the global archive and the obtained archive in the interval. The intervals which do not survive the selection mechanism are not considered anymore for the rest of the computations. In fact, after a certain number of depths, many intervals are not selected anymore and the number of required processors reduces until the results are not being improved and the algorithm stops automatically.

This selection mechanism and the above mentioned criterion for removing an interval from the list of intervals may

cause an undesired effect of producing gaps along the obtained approximated front. The phenomenon is observed in the subdivision method [18] where the front is reconstructed by applying recovering methods to the obtained front at the end.

4. FOCUSING MULTI-OBJECTIVE OPTIMIZATION

SIPO is considered to be a general framework for self-organized parallel optimization. In this paper, in order to illustrate its functionality, we work on population-based methods to solve multi-objective problems. While the goal in many of the multi-objective algorithms is to find a set of optimal solutions with good diversity and convergence on the true Pareto-front, the so called preference-based optimization techniques or guided optimization methods are particularly designed to focus on desired and preferred areas defined by the user. These kinds of optimization methods are especially suitable for optimization in SIPO as the optimization must focus inside some given interval. Recently, a categorization of existing approaches for different preference specifications required by the user has been given in [3, 20]. Since it is straight forward to guide a population of solutions in the objective space by Particle Swarm Optimization (PSO) techniques, e.g. [16, 17, 22], we select PSO as a possible approach for optimization in SIPO.

In Multi-Objective PSO (MOPSO), a set of N particles are considered as a population P_t in generation t. Each particle i has a position and velocity defined by $\vec{x}_i$ and $\vec{v}_i$ in the search space $S \in \mathbb{R}^n$. In generation $t+1$, a new velocity and position for each particle i is generated by updating the old ones as follows:

$$\begin{aligned} v_i^{j,t+1} &= W v_i^{j,t} + c_1 R_1 (P_i^{j,t} - x_i^{j,t}) + c_2 R_2 (P_g^{j,t} - x_i^{j,t}) \\ x_i^{j,t+1} &= x_i^{j,t} + v_i^{j,t+1} \end{aligned} \quad (1)$$

where $j = 1, \cdots, n$, W is called the inertia weight of the particle, c_1 and c_2 are two positive constants which we set to one, and R_1 and R_2 are random values in the range $[0, 1]$. In Equation (1), $\vec{P}_i^t$ denotes the best position which the particle i has obtained so far and is updated in every iteration. $\vec{P}_g^t$ denotes the position of the so called global best particle which is typically selected from a set of non-dominated solution. Since the global best particle has a great impact on the diversity of solutions in MOPSO, we use it in the following to focus on an interval. For a more extensive treatment of MOPSO techniques in general, the reader is referred to [17].

Imagine a population of particles is supposed to focus on an interval $[a, b]$ in the objective space (Figure 2). We select the non-dominated particle closest to the middle of the interval as the only global best particle $\vec{P}_g^t$ for all other particles in the population and denote it as G^t. The role of this global best particle is to guide all the particles towards the middle of the interval. In order to increase the impact of this solution, we change Equation 1 to:

$$\begin{aligned} v_i^{j,t+1} &= W v_i^{j,t} + R_1 (1 - W_f)(P_i^{j,t} - x_i^{j,t}) \\ &+ R_2 W_f (G^{j,t} - x_i^{j,t}) \end{aligned} \quad (2)$$

where $0 \leq W_f \leq 1$ and is called focusing factor. If we set it to one, G^t will influence the population to exploit the region around it. The lower this value, the higher is the impact of the personal best P_i^t.

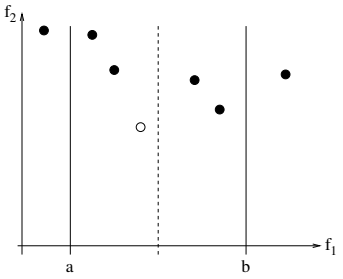


Figure 2: Focusing MOPSO inside the interval $[a, b]$: The non-dominated solution closest to the middle of the interval (white particle) is selected as the only global best particle for the population members (black particles).

In order to avoid stagnation in local optima, we usually employ a turbulence factor [15]. This works by randomly changing the positions of a given percentage of the particles in the parameter space. In our case, if we apply the turbulence factor to the particles in the entire parameter space, the solutions easily go out of the desired interval. As we want to focus on an interval, we only allow a percentage of particles to randomly move in a fixed area around the selected global best. In this way, we force a percentage of particles indicated by a **local search** factor to perform a local search by unified random sampling around the global best. This parameter will be tested in our experiments.

Algorithm 2: Focusing MOPSO

Input: $I := [I_{min}, I_{max}]$ interval
Output: a local archive

$a^0 := []$ empty local archive
$a^0 = AccessGlobalArchive$
for $i = 1$ *to* $PopSize$ **do**
  $pop^0(i) = Initialize(x_i^0, v_i^0, P_i^0)$
end
for $t = 1$ *to iterations* **do**
  $G^t = FindGlobalBest(pop^{t-1}, a^{t-1}, I)$
  $(pop^t, a^t) = Update(pop^{t-1}, G^t)$
  $(pop^t, a^t) = LocalSearch(G^t, LS)$
end

Algorithm 2 briefly explains Focusing MOPSO. The operations are employed for all the dimensions j which are not illustrated in the Algorithm. The goal of this algorithm is to create a local archive a which is first set equal to the global archive A (in the shared memory) by the function $AccessGlobalArchive$ and then updated. Within a given interval I, the algorithm finds a global best solution G^t which is an input to the Function $Update(pop^{t-1}, G^t)$. This function updates the positions and velocities of the particles in the population pop^{t-1} according to Equation 2. The non-dominated solutions are stored in the local archive a^t. The positions of the personal best particles are computed in this function which saves the most recent non-dominated position that a particle could obtain so far. The Function $LocalSearch(G^t, LS)$ performs a local search in a fixed area around the global best G^t. LS is the local search factor which is a percentage indicating the number of particles used

for the local search. Hereby, the population and the archive are updated again.

5. EXPERIMENTS

We select three different test problems containing convex, concave and disconnected fronts. Due to space limit in this paper, we only report the results on the well-known ZDT1, 2 and 3 functions [9]. As in SIPO the quality of solutions gets more precise over the division process, the optimization algorithm in each interval must only find a rough approximation of solutions. Therefore, Focusing MOPSO (F-MOPSO) is run for 20 particles and 20 iterations in all the experiments. The maximal global archive size is kept to 100 by a clustering mechanism [15]. We select the standard value 0.4 for inertia weight. All the experiments are repeated for 30 different runs. The starting interval is selected as $[0, 1]$ along the first objective for all the test problems.

In the parallel environment, we suppose that the processors have the same properties (in terms of memory and computing power)[2]. The communication overhead between the resources is considered to be negligible. The access to the global archive is achieved by using a shared memory. We start the computation from one and two computing resource and set the $NumofDivisions$ to 2. Since we work on a simulation of the parallel environment, the job assignment strategy is not being studied here.

For evaluations, we compute the average number of function evaluations and the quality of solutions based on the hypervolume metric (smetric in [23]). We set the reference point to $(3, 3)$. In addition, we measure the average number of required resources over different depths in the algorithm.

In SIPO, the number of required resources is estimated based on the quality of obtained solutions during the optimization. As a consequence, SIPO does not require an explicit stopping criteria, but the number of evaluations varies for different runs and settings which makes it difficult to compare two different sets of solutions. Therefore, we compare the results by looking at the quality (hypervolume) and the corresponding number of required evaluations. It is obvious, that if we have more function evaluations, we obtain better results with higher values of hypervolume. The best setting must have the smallest number of evaluations and the largest hypervolume.

5.1 Results

Table 1 shows the results of the first experiments. HV and SE_{HV} indicate the average hypervolume and the corresponding standard error. $\#Ev$ and SE_{Ev} show the average number of required evaluations and the standard error. $\#AMP$ is the average maximum number of processors required for the entire optimization process whereas $\#MAP_D$ indicates the maximum of the average number of processors required in one depth.

In the following, we first find the best parameter settings for F-MOPSO and then analyze different aspects of parallelization. The first experiments are carried out to determine the impact of the focusing factor W_f. The role of the global best particle is to guide the particles in the middle of a given interval. All of the experiments for ZDT1 and ZDT3 show that $W_f = 1.0$ gives us the best average HV with similar or even better results in terms of number of evaluations $\#Ev$ and number of intervals $\#AMP$ and $\#MAP_D$. For ZDT2, $W_f = 1.0$ gives us also the best average HV, whereas we require more $\#Ev$ (average of 200) than for $W_f = 0.9$. However, these results clearly show that the focusing factor in our approach must be the highest. The local search factor LS in F-MOPSO has a great impact on the solutions as we need to focus inside the intervals and additionally avoid stagnation in local optima. We select a fixed area of 0.2 around the global best particle for performing the local search for all the test problems. This value is set according to our preliminary experiments. Table 1 shows the results for different values of LS. From the results in Table 1, we observe that SIPOs with one starting processor outperform all SIPOs with 2 starting processors for all the LS values, as the solutions with $\#SP = 1$ clearly require less $\#Ev$ for the same HV than those with $\#SP = 2$. Concerning the focusing factor W_f, we conclude, as discussed before,

[2]This factor does not have an impact in our approach.

Table 1: Results of SIPO for different focusing factor W_f, number of starting processors $\#SP$ and local search factors LS on ZDT1, ZDT2 and ZDT3 test problems

W_f	$\#SP$	LS	HV	SE_{HV}	$\#Ev$	SE_{Ev}	$\#AMP$	$\#MAP_D$
ZDT1								
1.0	1	0.4	8.2441	0.0467	16380	729	39.0	12.4
1.0	2	0.4	8.4072	0.0374	26544	1118	63.2	20.4
1.0	1	0.3	8.3137	0.0473	16968	679	40.4	13.3
1.0	2	0.3	8.3680	0.0437	28728	1044	68.4	23.3
1.0	**1**	**0.2**	**8.3781**	**0.0489**	**17836**	**859**	**42.5**	**13.1**
1.0	2	0.2	8.3604	0.0452	26096	1179	62.1	21.7
1.0	1	0.1	8.2643	0.0580	19992	1097	47.6	13.9
1.0	2	0.1	8.2720	0.0445	27384	1519	65.2	20.3
0.9	1	0.4	8.0218	0.0611	16520	710	39.3	12.5
0.9	2	0.4	8.2065	0.0435	26656	1198	63.5	19.5
0.9	1	0.3	8.0985	0.0571	17668	733	42.1	12.7
0.9	2	0.3	8.0402	0.0452	26684	1407	63.5	20.9
0.9	1	0.2	7.9798	0.0660	20552	957	48.9	14.9
0.9	2	0.2	8.0190	0.0490	24360	1440	58.0	17.0
0.9	1	0.1	7.7029	0.0843	19992	1270	47.6	14.8
0.9	2	0.1	7.7727	0.0633	26124	1486	62.2	17.1
ZDT2								
1.0	1	0.4	7.2479	0.1810	10304	935	24.5	8.6
1.0	2	0.4	7.6307	0.1675	13328	1215	31.7	11.5
1.0	**1**	**0.3**	**7.6120**	**0.1428**	**10276**	**810**	**24.5**	**8.4**
1.0	2	0.3	7.4506	0.1595	12572	1015	29.9	9.0
1.0	1	0.2	6.9770	0.3010	9632	1029	22.9	8.9
1.0	2	0.2	7.4152	0.1778	13272	1304	31.6	11.0
1.0	1	0.1	6.9374	0.3056	10612	1143	25.3	8.6
1.0	2	0.1	6.8399	0.2611	10696	1195	25.5	8.8
0.9	1	0.4	6.4351	0.2489	7756	959	18.5	5.0
0.9	2	0.4	6.8238	0.1460	11032	1174	26.3	5.9
0.9	1	0.3	6.2715	0.1560	6608	732	15.7	4.1
0.9	2	0.3	6.6565	0.1627	9632	1099	22.9	5.2
0.9	1	0.2	6.0436	0.1786	6188	710	14.7	3.6
0.9	2	0.2	6.3902	0.1811	8596	773	20.5	4.3
0.9	1	0.1	4.9071	0.3719	5096	796	12.1	2.7
0.9	2	0.1	5.7303	0.2340	6496	765	15.5	4.0
ZDT3								
1.0	1	0.4	9.1450	0.1056	13076	488	31.1	9.7
1.0	2	0.4	9.4120	0.0755	15148	638	36.1	10.1
1.0	**1**	**0.3**	**9.3122**	**0.0891**	**13216**	**577**	**31.5**	**9.4**
1.0	2	0.3	9.5139	0.0757	16828	638	40.1	11.4
1.0	1	0.2	9.2614	0.0769	13384	525	31.9	9.5
1.0	2	0.2	9.3338	0.0762	16156	667	38.5	10.5
1.0	1	0.1	9.0677	0.1060	15204	615	36.2	10.0
1.0	2	0.1	9.1793	0.1193	17948	921	42.7	12.1
0.9	1	0.4	8.8418	0.0934	13692	462	32.6	9.5
0.9	2	0.4	8.6382	0.1039	14588	852	34.7	9.9
0.9	1	0.3	8.8589	0.0888	13916	496	33.1	9.8
0.9	2	0.3	8.6723	0.0919	15820	927	37.7	9.7
0.9	1	0.2	8.4806	0.0944	14504	660	34.5	9.2
0.9	2	0.2	8.5982	0.1152	15540	950	37.0	9.4
0.9	1	0.1	8.0370	0.1333	14448	900	34.4	8.7
0.9	2	0.1	8.0830	0.1464	16100	1055	38.3	9.5

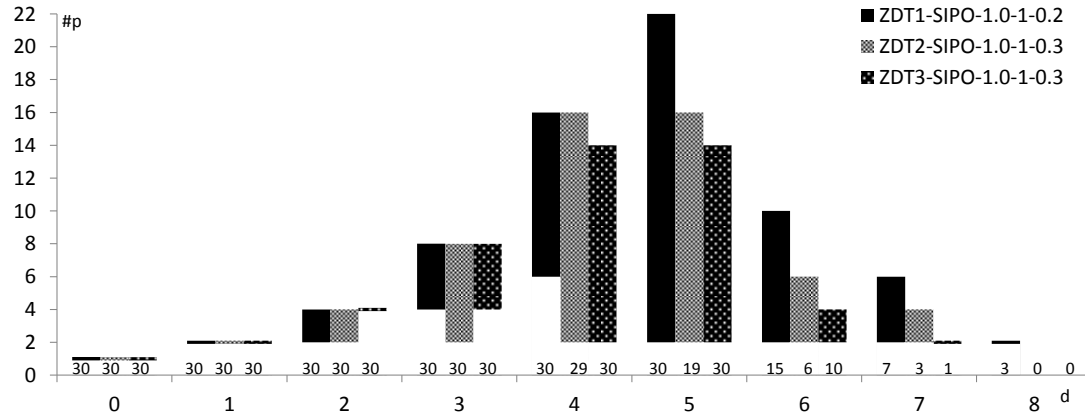


Figure 3: The average number of computing resources required by SIPO-1.0-1-0.2 and SIPO-1.0-1-0.3 in each depth is shown for ZDT1, ZDT2 and ZDT3. The columns show the minimum and the maximum number of processing units. The numbers below the columns describe the number of runs which reached the corresponding depth.

that SIPO-1.0-1 (SIPO-W_f-$\#SP$) has a larger hypervolume and almost the same number of function evaluations than SIPO-0.9-1 for all the test problems. Among the solutions with the same W_f and $\#SP$ values, the solutions obtained with local search factor (LS) of 0.2 have the best HV and $\#Ev$ for ZDT1, whereas 0.3 for ZDT2 and ZDT3. For the next experiments and comparisons, we take these best results as (SIPO-W_f-$\#SP$-LS) SIPO-1.0-1-0.2 for ZDT1 and SIPO-1.0-1-0.3 for ZDT2 and ZDT3.

In the following, we investigate the average maximum number of required processors $\#AMP$ and the maximum average number of resources at one depth $\#MAP_D$ by SIPO-1.0-1-0.2 for ZDT1 and SIPO-1.0-1-0.3 for ZDT2 and ZDT3 as shown in Table 1. We obtain $\#AMP$ values of 42.5, 24.5 and 31.5 and $\#MAP_D$ values of 13.1, 8.4 and 9.4 for ZDT1, ZDT2 and ZDT3, respectively. The number of resources at each depth illustrates the parallel resources required at the same time. This indicates that for our test problems, we require a maximum of 14 (13.1) processors in average, 22 processors the highest and 1 processor the lowest which is illustrated in Figure 3. Since we run SIPO for 30 different runs, we realize that not all of them reach the same depth. Some runs stop the optimization earlier than the others. Therefore, we indicate the number of runs which successfully reached a certain depth in Figure 3. We observe that in most of the cases, the intervals survive 5 successive divisions. The further divisions at depth 6 can be considered as fine corrections on the approximated Pareto-front.

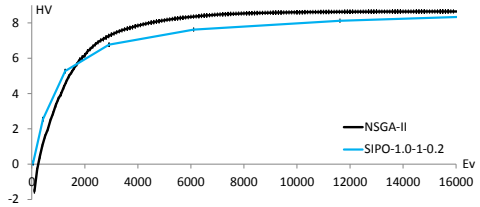
5.2 Comparison

For a more in-depth evaluation, we additionally compare the results of SIPO with a number of straight forward alternatives below. Based on these comparisons, one can compute the speed-up factors in parallelization. For these comparisons, we select the best obtained results from above, namely SIPO-1.0-1-0.2 for ZDT1 and SIPO-1.0-1-0.3 for ZDT2 and ZDT3.

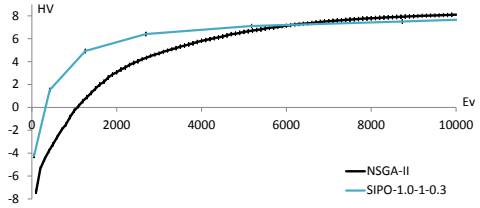
- **Case 1:** Instead of running a parallel algorithm, a simple alternative would just be to run a single multi-objective algorithm on one processor available for the same number of evaluations. While this scenario uses less computational power overall, it will tell us how much we can benefit from parallelization. We select a baseline algorithm which obtains the best results for these problems according to [8] and compare the algorithms in terms of convergence rate. We select the standard parameter setting for the baseline algorithm (NSGA-II) from [8] (population size 100, 200 generations, 0.9 probability of crossover with $\eta_c = 15$ and 0.033 probability of mutation with $\eta_m = 20$).

- **Case 2:** We take the maximum number of processors which SIPO requires in average for the optimization and use them in parallel for the same number of evaluations. We divide the main given interval in terms of the first objective. Each processor gets an interval dedicated by us for the optimization and runs F-MOPSO to focus on. By this experiment, we want to observe the influence of different depths on the quality of the results. We take $\#Ev$ from SIPO-1.0-1-0.2 / SIPO-1.0-1-0.3 (17836, 10276 and 13216 for ZDT1 to ZDT3) and divide that by $\#AMP$ (43, 25 and 32). In this way, we compute $\#Ev$ for each processor. The number of particles is kept to 20 and therefore the number of iterations is 21 for all the problems.

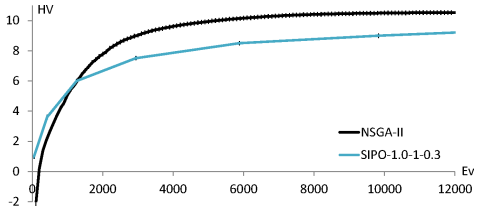
- **Case 3:** We take the maximum average number of processors which are run in parallel by SIPO, i.e., the maximum average number of processors required in one depth ($\#MAP_D$). Like in Case 2, we divide the given interval into sub-intervals and run the processors in parallel and altogether for the same number of evaluations. This experiment is meant to measure the quality of solutions, when we only have a certain number of processors available for a longer time than in Case 2. In more details, we take $\#Ev$ from SIPO-1.0-1-0.2 / SIPO-1.0-1-0.3 (17836, 10276 and 13216 for ZDT1 to ZDT3) and divide that by $\#MAP_D$ (13, 8 and 9). The number of iterations in each processor is



(a) ZDT1



(b) ZDT2



(c) ZDT3

Figure 4: Quality in terms of the average hypervolume of the best results obtained by SIPO in comparison with the baseline algorithm (NSGA-II) over the number of evaluations

computed as 69, 64 and 73 for ZDT1 to ZDT3 for 20 particles.

- **Case 4:** We take the average of the maximum number of parallel processors over all depths by SIPO. The average values are 8, 7 and 6 for the three test problems. Like in other cases, we divide the given interval into sub-intervals and run the processors in parallel for the same number of evaluations. In more details, we take $\#Ev$ from SIPO-1.0-1-0.2 / SIPO-1.0-1-0.3 (17836, 10276 and 13216 for ZDT1 to ZDT3) and divide that by 8, 7 and 6. The number of iterations in each processor is computed as 111, 73 and 110 for ZDT1 to ZDT3 for 20 particles.

Figures 4 (a)-(c) show the results of SIPO and Case 1 for ZDT1 to ZDT3 with the corresponding standard errors. The two methods have equally good and small standard errors. The plots indicate the average values of hypervolume over the number of evaluations. In these experiments, SIPO obtains for a small $\#Ev$ better solutions than NSGA-II for ZDT1 to ZDT3. NSGA-II clearly obtains very good solutions for higher $\#Ev$ and the convergence rate increases by increasing the amount of evaluations. However, SIPO has a faster convergence in the beginning which can be explained by the parallelization effect. SIPO has a stair-like growth in the convergence rate, as the global archive is updated after each depth. SIPO behaves worse than NSGA-II the more $\#Ev$ is increasing. We explain this by the circumstance, that after the first two depth some of the good intervals are omitted due to missing nondominated solutions in these intervals. This can specially be recognized by the ZDT3 problem, which contains gaps and therefore, SIPO has probably omitted some of the good intervals containing gaps. Surprising results are obtained for the ZDT2 problem. This problem is known to be one of the difficult concave multi-objective problems. The results with low $\#Ev$ values obtained by SIPO are much better than NSGA-II and with a very good convergence rate. This can also be explained by the parallelization effect.

Table 2: Results of SIPO, Case 2, Case 3 and Case 4 with $W_f = 1.0$ and LS of 0.2 (ZDT1) and 0.3 (ZDT2 and ZDT3)

	Test	HV	SE_{HV}	$\#Ev$	$\#AMP$
ZDT1	SIPO	8.3781	0.0489	17836(±859)	42.5 - 13.1
	Case 2	3.7178	0.0786	18060	43
	Case 3	7.4544	0.0431	17940	13
	Case 4	8.1703	0.0359	17760	8
ZDT2	SIPO	7.6120	0.1428	10276(±810)	24.5 - 8.4
	Case 2	3.4809	0.0829	10500	25
	Case 3	6.8323	0.0915	10240	8
	Case 4	7.0200	0.0713	10220	7
ZDT3	SIPO	9.3122	0.0891	13216(±577)	31.5 - 9.4
	Case 2	4.9334	0.0992	13440	32
	Case 3	8.7360	0.0598	13140	9
	Case 4	9.3449	0.0789	13200	6

Table 2 illustrates the results for Case 2, Case 3 and Case 4, where the results of Case 2 have the lowest quality. On the other hand, Case 4 with the lowest number of parallel processors obtains the best results among the cases. This indicates that we do not benefit from starting with a large number of processing units. Additionally, we observe that if we adaptively change the number of processing units on demand, the results are much better than when starting by a large number of processors. However, this effect can depend on the way we divided the objective space into intervals. Other methods like cone separation [5] could obtain better results. In our experiments, we want to show the effect of the self-organized optimization and do not intent to analyze the way the objective space is divided. SIPO obtains better quality of solutions than Case 4 for ZDT1 and ZDT2 problems where for ZDT3 Case 4 outperforms SIPO. This is due to the shape of the Pareto-front of the test problems. Both ZDT1 and ZDT2 have connected Pareto-fronts where ZDT3 has a disconnected front. For ZDT3 problem, we realize that some intervals are deleted automatically by SIPO during the division process which causes undesired gaps along the approximated front.

6. CONCLUSIONS AND FUTURE WORK

In this paper, we study a new approach called Self-organized Invasive Parallel Optimization (SIPO) for solving optimization problems using parallel platforms. One of the main aspects when using parallel environments is to determine the number of required processors to solve a certain problem. In our approach, we develop an algorithm which determines the number of required processors during the optimization based on the quality of the so far obtained solutions. This approach is particularly appropriate for interactive problems on systems like clouds or grids so that the users can stop the optimization anytime by looking at the quality of the results. For evaluating our approach, we select multi-objective optimization problems and perform the optimization by a PSO based focusing algorithm (F-MOPSO). The role of F-

MOPSO is to focus on certain given intervals. SIPO starts from an interval given by the user with one computing resource, which takes the role of a master. In the case that the interval of the master resource passes a certain selection mechanism, it is divided into smaller intervals which are assigned to other resources by the master. The resources run an F-MOPSO, update a global archive in the shared memory and take the role of a master. This process can successively be continued. The quality of solutions gets more precise over the successive divisions of the intervals. The selection mechanism determines whether an interval has to be further divided or not. After a certain number of divisions, SIPO stops automatically.

The results of our experiments show that for the selected test problems, we require a significantly smaller number of parallel resources than the total number of resources in our platform. The results are comparable with results of existing non-parallel approaches and much better than other parallel variants (with the same number of evaluations and different number of processors). However, in some cases we realize that some intervals are deleted causing undesired gaps along the approximated front. This will be investigated in our future work. In addition, we intend to extend our approach for dividing the objective space in terms of all the objectives. Furthermore, as SIPO is a general framework, we will employ other optimization algorithms in SIPO and investigate it on real-world problems. The efficiency of SIPO will be studied in comparison with other existing parallel approaches.

7. REFERENCES

[1] A. Abdulazim, F. Arifin, F. Hannig, and J. Teich. FPGA Implementation of an Invasive Computing Architecture. In *Proceedings of the IEEE International Conference on Field Programmable Technology (FPT)*, pages 135–142, Sydney, Australia, 2009. IEEE.

[2] D. Abramson, A. Lewis, and T. Peachy. Nimrod/o: A tool for automatic design optimization. In *The 4th International Conference on Algorithms and Architectures for Parallel Processing*, 2000.

[3] J. Branke. Consideration of partial user preferences in evolutionary multiobjective optimization. In *Multiobjective Optimization*, pages 157–178, 2008.

[4] J. Branke, T. Kaußler, and H. Schmeck. Guidance in evolutionary multi-objective optimization. *Advances in Engineering Software*, 32(6):499 – 507, 2001.

[5] J. Branke, H. Schmeck, K. Deb, and M. Reddy. Parallelizing Multi-Objective Evolutionary Algorithms: Cone Separation. In *IEEE Congress on Evolutionary Computation*, pages 1952–1957, 2004.

[6] Y. Censor and S. A. Zenios. *Parallel Optimization: Theory, Algorithms, and Applications*. Oxford University Press, New York/Oxford, 1997.

[7] K. Deb. *Multi-objective Optimization using Evolutionary Algorithms*. WILEY, 2002.

[8] K. Deb, A. Pratap, and S. Agarwal. A fast and elitist multi-objective genetic algorithm: Nsga-ii. *IEEE Trans. on Evolutionary Computation*, 6(8), 2002.

[9] K. Deb, L. Thiele, M. Laumanns, and E. Zitzler. Scalable multi-objective optimization test problems. In *Congress on Evolutionary Computation*, volume 1, pages 825–830, 2002.

[10] K. Deb, P. Zope, and A. Jain. Distributed computing of pareto-optimal solutions with evolutionary algorithms. In *International Conference on Evolutionary Multi-Criterion Optimization*, pages 534–549, 2003.

[11] D. Gelernter. Multiple tuple spaces in linda. In *PARLE 89, Vol. II: Parallel Languages*, pages 20–27, 1989.

[12] P. Møller-Nielsen and J. Staunstrup. Problem-heap: A paradigm for multiprocesor algorithms. *Parallel Computing*, 4(1):63 – 74, 1987.

[13] S. Mostaghim, J. Branke, and H. Schmeck. Multi-objective particle swarm optimization on computer grids. In *The Genetic and Evolutionary Computation Conference (GECCO)*, volume 1, pages 869–875, 2007.

[14] S. Mostaghim and H. Schmeck. Self-organized parallel cooperation for solving optimization problems. In *22nd International Conference on Architecture of Computing Systems*, volume 5455 of *Lecture Notes in Computer Science*, pages 135–145, Berlin, Heidelberg, 2009. Springer.

[15] S. Mostaghim and J. Teich. Strategies for finding good local guides in multi-objective particle swarm optimization. In *IEEE Swarm Intelligence Symposium*, pages 26–33, 2003.

[16] S. Mostaghim, H. Trautmann, and O. Mersmann. Preference-based multi-objective particle swarm optimization using desirabilities. In *Parallel Problem Solving from Nature (PPSN)*, pages 101–110. Parallel Problem Solving from Nature (PPSN), Springer, 2010.

[17] M. Reyes-Sierra and C. A. Coello Coello. Multi-objective particle swarm optimizers: A survey of the state-of-the-art. *International Journal of Computational Intelligence Research*, 2(3):287–308, 2006.

[18] O. Schütze, S. Mostaghim, M. Dellnitz, and J. Teich. Covering Pareto sets by multilevel evolutionary subdivision techniques. In *Proceedings of Second International Conference on Evolutionary Multi-Criterion Optimization*, pages 118–132, 2003.

[19] E.-G. Talbi, S. Mostaghim, T. Okabe, H. Ichibushi, G. Rudolph, and C. A. Coello Coello. *Parallel Approaches for Multiobjective Optimization*, pages 349–372. Springer Verlag, 2008.

[20] H. Trautmann and J. Mehnen. Preference-Based Pareto-Optimization in Certain and Noisy Environments. *Engineering Optimization*, 41:23–38, 2009.

[21] D. A. V. Veldhuizen, J. Zydallis, and G. B. Lamont. Considerations in engineering parallel multiobjective evolutionary algorithms. In *IEEE Transactions on Evolutionary Computation, Vol. 7, No. 2*, pages 144–173, 2003.

[22] U. K. Wickramasinghe and X. Li. Integrating user preferences with particle swarms for multi-objective optimization. In *Proceedings of the conference on Genetic and evolutionary computation (GECCO)*, pages 745–752, 2008.

[23] E. Zitzler. *Evolutionary Algorithms for Multiobjective Optimization: Methods and Applications*. Shaker, 1999.