On the Performance of Master-Slave Parallelization Methods for Multi-Objective Evolutionary Algorithms ⋆

Alexandru-Ciprian Zăvoianu[1,3], Edwin Lughofer[1], Werner Koppelstätter[3,4], Günther Weidenholzer[3,4], Wolfgang Amrhein[2,3], Erich Peter Klement[1,3]

[1] Department of Knowledge-based Mathematical Systems/Fuzzy Logic Laboratory Linz-Hagenberg, Johannes Kepler University of Linz, Austria
[2] Institute for Electrical Drives and Power Electronics, Johannes Kepler University of Linz, Austria
[3] ACCM, Austrian Center of Competence in Mechatronics, Linz, Austria
[4] LCM, Linz Center of Mechatronics, Linz, Austria

Abstract. This paper is focused on a comparative analysis of the performance of two master-slave parallelization methods, the basic *generational* scheme and the *steady-state asynchronous* scheme. Both can be used to improve the convergence speed of multi-objective evolutionary algorithms (MOEAs) that rely on time-intensive fitness evaluation functions. The importance of this work stems from the fact that a correct choice for one or the other parallelization method can lead to considerable speed improvements with regards to the overall duration of the optimization. Our main aim is to provide practitioners of MOEAs with a simple but effective method of deciding which master-slave parallelization option is better when dealing with a time-constrained optimization process.
Keywords: evolutionary computation, multi-objective optimization, performance comparison, master-slave parallelization, steady-state evolution

1 Introduction and State-of-the-art

Many real-world optimization problems are multi-criteria by nature and usually involve several conflicting objectives (e.g. cost vs. quality, risk vs. return on investment). Problems falling within this class are referred to as *multi-objective optimization problems* (MOOPs in short) and, generally, such problems do not have a single solution. Solving them requires finding a set of non-dominated solutions called the *Pareto-optimal set*. Each solution in this set is better than any other solution in the set with regards to at least one optimization objective (i.e., it is not fully dominated by another solution). The objective space representation of the Pareto-optimal set is called the *Pareto front*.

Multi-objective evolutionary algorithms (MOEAs) have proven to be one of the most successful soft computing techniques for solving MOOPs [2]. This is because of their inherent ability to produce complete Pareto-optimal sets over single runs. As most

⋆ This work was conducted in the realm of the research program at the Austrian Center of Competence in Mechatronics (ACCM), which is a part of the COMET K2 program of the Austrian government. The work-related projects are kindly supported by the Austrian government, the Upper Austrian government and the Johannes Kepler University Linz. The authors thank all involved partners for their support. This publication reflects only the authors' views.

stochastic methods, MOEAs are approximate methods that cannot guarantee finding the strictly optimal solution set (i.e., the *true Pareto front* of the problem), but they are fairly flexible and robust and can find high quality non-dominated solution sets in a reasonable amount of time.

The main drawback of MOEAs is the fact that they require a large number of solutions to be evaluated during an optimization run. The issue can become particularly problematic for optimization problems that require very time intensive fitness evaluation functions in order to compute objective or constraint values. In such cases, optimization runs can last for several days, see for instance the approach in [13] where MOEAs are used for the optimization of combustion in a diesel engine and the approach in [17] applied for optimizing design parameters of electrical drives.

This problem of very lengthy optimization runs is usually alleviated by the parallelization and distribution of the MOEA over a computer cluster or grid environment. There are several paradigms (architectural and/or conceptual models) of parallelizing a MOEA: master-slave, island, diffusion, hierarchical and hybrid models (please see Chapter 8 of [1]). The most straight forward and easiest to implement parallelization method for evolutionary algorithms is the master-slave model: fitness evaluations are distributed between several slave nodes, while all the evolutionary operations (selection, crossover, mutation) are performed on a master node. The master-slave parallelization is suitable both for a classic, generational approach, as well as for an asynchronous parallelization approach similar to the steady-state selection scheme [10].

In several real-world master-slave parallelization setups for MOEAs, the duration of the sequential computation tasks is also very important. This usually happens because of some rather lengthy pre-evaluation steps that must be performed locally on the master node for each generated individual, before dispatching the individual for remote fitness evaluation on the slave nodes. The reasons for performing these pre-evaluation steps on the master node may include security concerns, software licensing issues, network configuration settings, etc. It is fairly intuitive that whenever the average duration of the sequential task carried out on the master node is significant with regards to the average duration of the fitness evaluation task, the speed-up that can be achieved by using a parallel / distributed hardware architecture is affected (q.v. Amdahl's law).

A recent study indicates that, for optimization problems with a heterogeneous (non-constant) time-wise fitness distribution, the steady state asynchronous parallelization is somewhat better in terms of convergence (Pareto quality and global run-time) than the generational approach [13]. In [8], Durillo et al. also show evidence that applying a steady state approach can bring improvements in terms of Pareto quality. The present research builds on these earlier findings and tries to determine the reasons that might influence the average performance of the two master-slave parallelization schemes in the context of MOEAs. Our main intention is to help practitioners in this field to decide what is the most efficient parallelization option based on the particularities of their concrete optimization scenarios. The comparison is especially focused on the practical aspect of Pareto quality / run-time performance: *which parallelization method is more likely to deliver the highest quality solution in a pre-defined global run-time interval*.

The tests we report on were performed taking into consideration two of the most widely used MOEAs: the Non-dominated Sorting Genetic Algorithm II (NSGA-II) [4]

and the Strength Pareto Evolutionary Algorithm 2 (SPEA2) [15]. At a high level of abstraction, NSGA-II and SPEA2 are in fact different MOOP orientated implementations of the same concept - the $(\mu + \lambda)$ evolutionary strategy (where μ denotes the archive size and λ the population size). The two main features of both algorithms are *a*) a highly elitist approach that is based on an archive population that stores the best individuals found during the run; *b*) a two-tier selection for survival function that uses a primary Pareto non-dominance metric and a secondary solution density estimation metric.

2 The Considered Master-Slave Parallelization Schemes

The diagram in Figure 1 provides a general explanation of the computation cycles used in both master-slave parallelization schemes. The basic principles of the two paralleliza-

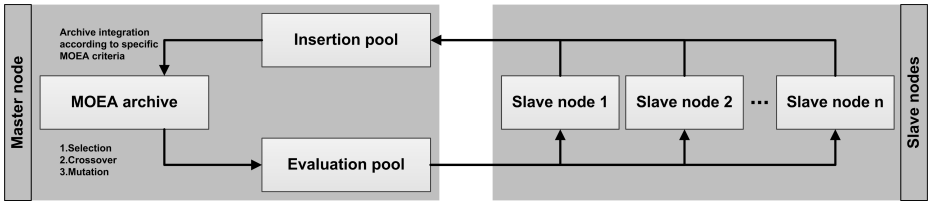

Fig. 1. Diagram of the GEN-MSPS and SSA-MSPS computation cycle

tion schemes for a generic $(\mu + \lambda)$-archiving based MOEA are:

1. **The Generational Master-Slave Parallelization Scheme (GEN-MSPS)** - In this case, the computation cycle is regulated by a **synchronization step**. This step occurs at the integration of individuals from the *Insertion pool* into the archive. The master node must block until all the λ individuals of the current population have been evaluated on the slave nodes. After this requirement is satisfied, the specific $(\mu + \lambda)$-archiving algorithm is used in order to update the *MOEA archive* on the master node. Afterwards, all the λ individuals of the next generation are created sequentially on the master node using the specific genetic operators (selection, recombination and mutation) and inserted into the *Evaluation pool*. Each slave node asynchronously selects an individual from this pool, evaluates it and afterwards places the results in the *Insertion pool*. The procedure described above repeats itself until the optimization stopping criterion is met. From an algorithmic point of view, this parallelization scheme is identical to a sequential implementation.
2. **The Steady-State Asynchronous Master-Slave Parallelization Scheme (SSA-MSPS)** - In this case, the computation cycle is regulated only by the interplay between the computation time requirements of the different parts of the algorithm (i.e. fitness evaluation, generation of new individuals, archive update, etc.). The slave nodes operate in the same way as in the generational parallelization scheme. The master node operates based on a very simple loop. While the stopping criterion is not met, the master node first checks if there is an evaluated individual in the

insertion pool and if such an individual exists, it collects it and updates the *MOEA archive*. This is the main difference to GEN-MSPS, which in each cycle has to wait for the evaluation of *all* individuals before new individuals can be generated. Secondly, the master node creates one new individual and inserts it immediately into the *Evaluation pool*. The above computation cycle resembles classical steady-state selection as, at a given time, one new individual is created and one evaluated individual is collected and, if fit enough, inserted into the archive. It is noteworthy that the SSA-parallelization scheme changes the algorithmic behavior of the given MOEA to that of an asynchronous $(\mu + 1)$-evolutionary strategy.

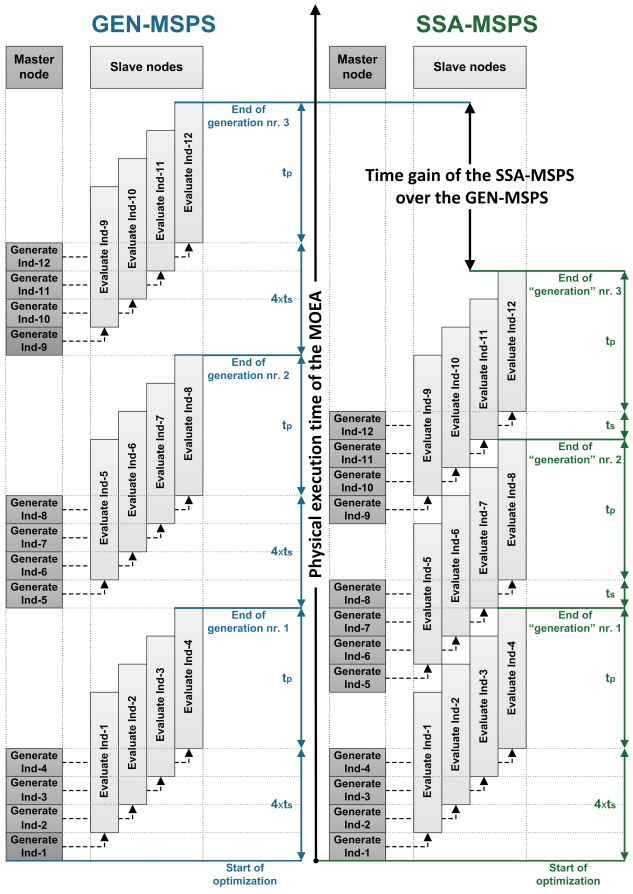

Fig. 2. The comparative computation steps of GEN-MSPS and SSA-MSPS for 3 generations of size 4 in a distributed computing environment with one master node and 4 slave nodes

Figure 2 provides an example of how individuals are processed by the two parallelization methods. The lack of a synchronization step in SSA-MSPS enables this

method to evaluate more individuals per time interval than GEN-MSPS. We shall investigate this quantitative aspect in Section 3. The negative side to using SSA-MSPS is that the same lack of generational synchronization is expected to make SSA-MSPS achieve worse results than GEN-MSPS in terms of Pareto front quality after evaluating a fixed number of individuals. This qualitative aspect is treated in Section 4. The last part of Section 4 contains an interpretation of the interplay between the quantitative and qualitative observations.

3 Examining the Quantitative Performance

3.1 The Basic Model

As mentioned in the previous section, it makes sense to presume that, given the same hardware architecture and specific MOEA settings, SSA-MSPS is able to compute faster than GEN-MSPS a given number of individuals (please see Figure 2). Another way of looking at this is that, by using the SSA-MSPS, one will be able to create and evaluate more individuals in the same time interval. We shall now attempt to quantify this improvement and to evaluate how the interplay between the duration of the parallel fitness evaluations and the duration of the synchronous tasks affects it.

Our theoretical model consists of a $(\mu + \lambda)$-archiving MOEA that is distributed over a computing environment with λ slave nodes. We mark with $t_p > 0$ the duration (in time units) of distributing and performing the fitness evaluation of any individual on any slave node. We also mark with $t_s > 0$ the cumulative duration of the sequential computation tasks (i.e., genetic operations + possible pre-evaluation tasks) that are performed on the master node in order to create one individual. In this section we assume that t_s and t_p are constant. We define the *parallelization ratio* as:

$$r = \lceil \frac{t_p}{t_s} \rceil \tag{1}$$

When considering the GEN-MSPS approach, it is quite straightforward that when it is using more than $r+1$ slave nodes simultaneously, the computation time is not further improved. As such, the following reasoning is made under the restriction $r+1 \geq \lambda$.

Assuming that other miscellaneous computation times are negligible with regards to (or integrated in) t_s and t_p, the total time required to compute any generation of λ individuals using the GEN-MSPS is $(\lambda \times t_s) + t_p$. In case of the SSA-MSPS, the time required to compute the first λ individuals is also $(\lambda \times t_s) + t_p$, but the time required to compute any of the next batches of λ individuals is $(t_s + t_p)$, as sketched in Figure 2.

As such, under the assumptions of our theoretical model, when wishing to compute N generations, the overall computation time is *a*) $(\lambda \times t_s + t_p) \times N$ in case of the GEN-MSPS scheme; *b*) $(\lambda \times t_s + t_p) + (t_s + t_p) \times (N-1)$ in case of the SSA-MSPS scheme. After equalizing these computation times and performing the necessary calculations, we obtain that in the time interval required by the GEN-parallelization to compute N generations of λ individuals, the SSA-parallelization can compute $\Delta_{struct}\%$ more individuals, where Δ_{struct} is given by:

$$\Delta_{struct} = \frac{(N-1)(\lambda-1) \times t_s}{N \times (t_s + t_p)} \times 100 \tag{2}$$

We shall refer to this measure as the **structural improvement** that SSA-MSPS has over GEN-MSPS in terms of computed individuals per given time interval.

It is important to note that while Δ_{struct} does depend on the number of generations, the population size and also on the order of size between these two variables and t_s, the dominant factor that influences Δ_{struct} is the ratio between t_s and t_p, or in other words, the parallelization ratio r. As we fix $\lambda = 100$, $N = 500$, and $t_s = 1$, by varying the value of t_p, we obtain the dependency of Δ_{struct} on r. The plot is presented in Figure 3 - basic model curve. Unsurprisingly, it shows that the quantitative improvement that SSA-MSPS brings, decreases exponentially with regards to the parallelization ratio.

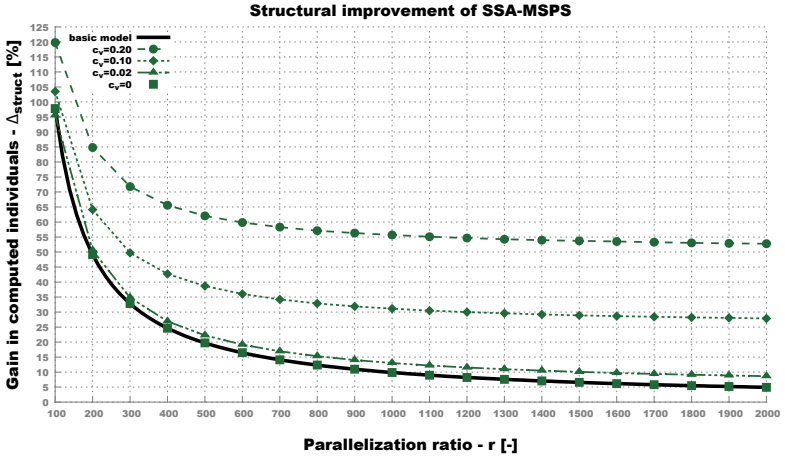

Fig. 3. Δ_{struct} curves for different parallelization ratios and different degrees of variance (i.e. c_v) in the time-wise distribution of the fitness evaluation function

Although valuable in establishing a baseline for the comparison between the GEN-parallelization and the SSA-parallelization schemes, the above described comparison has one severe limitation: it is strongly influenced by the idealistic assumption that the duration of the fitness evaluation task is constant (i.e. there is zero variance in the time-wise distribution of the fitness evaluation function). In the next subsection, we proceed to address this issue in order to enhance our quantitative performance model.

3.2 The Effect of Variance on the Quantitative Performance

Initially, we tested the theoretical model proposed in the previous section. Using a constant (i.e. variance free) time-wise fitness distribution, we simulated the time required by GEN-MSPS and SSA-MSPS runs when having various values of the coefficient of parallelization (1). The obtained results (Figure 3 - the $c_v = 0$ data points) confirm the Δ_{struct} behavior indicated by the theoretical model from (2).

Next, we evaluated the influence of having various degrees of variance in the time-wise distribution. In these tests, the fitness evaluation of each individual took t_p millisec-

onds where $t_p \sim \mathcal{N}(m, \sigma)$. As we fixed $t_s = 1$, in this case, $r \sim m$ and, when scaling up m we also modified σ in order to keep the coefficient of variation, $c_v = \frac{\sigma}{m}$, constant at a preset value. The maximum amount of variance that we could consider, under a normal distribution assumption, was given by $c_v = 0.2$. Because of the induced stochasticity, for each value of r we performed 100 tests and we report averaged results.

The plot in Figure 3 shows how Δ_{struct} behaves when using four different c_v values. The curves clearly show that the logarithmic decrease of Δ_{struct} is less intense with increased variance. Further experiments have also shown that, with variance in the time-wise fitness distribution function, after reaching a lower threshold, the value of Δ_{struct} tends to stabilize. We have run simulations up to $r = 10^7$ with a step size of 10000 and, in Table 1, we report the discovered lower thresholds of Δ_{struct} for different variance levels. We mention that, in the absence of variance, for $r = 10^7$, $\Delta_{struct} = 0.000988\%$.

The conclusion of the above tests is that the theoretical model (2) gives an accurate lower limit for Δ_{struct} but the value of Δ_{struct} for a given parallelization ratio r is significantly higher when having variance in the time-wise fitness distribution. Furthermore, in the presence of variance, Δ_{struct} is lower bounded by variance-specific thresholds that display a remarkable stability even at very high values of r.

Table 1. The observed variance-specific lower thresholds of Δ_{struct}

c_v [-]	Lower threshold for Δ_{struct} [%]	Δ_{struct} for $r = 10^7$ [%]
0.20	50.0822	50.2134
0.10	25.0858	25.1224
0.05	12.5940	12.6135
0.02	5.0969	5.1017

4 Examining the Qualitative Performance - Empirical Results

4.1 Evaluation Framework Setup

The qualitative performance of the two considered master-slave parallelization schemes depends on the concrete MOOP to be solved, on the used MOEA and on the parameterization of the algorithm. In the following paragraphs we describe the details of the performance evaluation framework we used.

Test Problems We have chosen for benchmarking purposes four standard, artificial, test problems from evolutionary multi-objective literature. Our choice of artificial problems is self-evident as it is very helpful to know the ground truth (i.e., the true Pareto front) in order to compare between parallelization results. The problems have been specially selected as they propose different degrees of difficulty and different convergence behaviors for the two MOEA algorithms that we experiment with. The MOOPs we

consider are *a*) **KSW** - a classic optimization problem with 10 variables and two objectives based on Kursawe's function [11]; *b*) **DTLZ7** - a problem with 22 variables and 3 objectives that is aimed at testing the performance of a MOEA on discontinuous Pareto fronts [5]; *c*) **ZDT6** - a problem with 10 variables and 2 objectives that proposes difficulties regarding the non-uniformity of the search space [14]; *d*) **LZ09-F1** - a problem with 30 variables and two objectives part of the LZ09 problem set [12] which is particularly difficult for classic MOEAs like NSGA-II and SPEA2.

The computation of the fitness values for all four problems is very fast on any modern processor. In order to make the MOEAs exhibit the desired test behavior, the fitness computation times were artificially increased (as described in Section 3.2).

MOEA Parameterization Our implementation of the NSGA-II and SPEA2 algorithms is loosely based on the one provided by the jMetal package [7]. Because our main goal is to compare the performance of two different parallelization models on the same algorithm, we use, more or less, standard parameterization options.

As such, for all the performed tests, we used a population size of 100 individuals and an archive size of 100. In the case of SSA-MSPS the term "generation" is used to denote a batch of 100 individuals. We used three standard genetic operators from MOEA literature: binary tournament selection, simulated binary crossover [3] and polynomial mutation. We used standard values to parameterize these genetic operators: 0.9 for the crossover probability, 20 for the crossover distribution index, $1/L$ for the mutation probability (where L is the number of variables) and 20 for the mutation distribution index.

We make our comparisons based on rather short runs of 500 generations (50.000 individuals). This is because after 500 generations we reach generally good solutions for all the considered test problems and because we are particularly interested in studying the middle-stage convergence behavior of GEN-MSPS and SSA-MSPS. We consider that the behavior of the two parallelization schemes in the late-stage of convergence is of less practical importance for us. This is because time constraints are very important in many real-world industrial optimization scenarios and practitioners rarely run an optimization for more than a few hundred generations. If one particular method is constantly outperforming the other in a late-stage of convergence, given the descriptions from Section 2, one can easily switch during the optimization to the best performing parallelization method by enabling or disabling the mentioned **synchronization step**.

Quality Indicators For a given solution set S, the hypervolume associated with this solution set, $\mathscr{H}(S)$, has the advantage that it is the only Pareto front quality estimation metric for which there is a theoretical proof [9] of a monotonic behavior. This means that the maximization of the hypervolume constitutes the necessary and sufficient condition for the set of solutions to be *maximally diverse Pareto optimal solutions of a discrete, multi-objective, optimization problem* [9]. In light of this, for any optimization problem, the *true Pareto front* has the highest achievable hypervolume value.

In our case, for a given MOOP, the monotonic property of the hypervolume metric makes it ideal for assessing the relative quality of an arbitrary solution set S_*. Let us mark with S_{true} the *true Pareto front* of our MOOP. As we deal with artificial problems where S_{true} is known, we can present the quality of the given solution set as a percentile

obtained by reporting the hypervolume measure of this solution set to the hypervolume value associated with the true Pareto front of the given MOOP:

$$qual(S_*) = \frac{\mathscr{H}(S_*)}{\mathscr{H}(S_{true})} \times 100 \tag{3}$$

Expressing the quality of a solution set as a true hypervolume percent also enables us to define more accurately what we mean by the syntagms early, middle and late-stage of convergence. For the purpose of this research, in light of the motivations presented in the last paragraph of the previous section, we define a MOEA as being in the early stage of convergence if $qual(A_*) \leq 15$ where A_* denotes the current archive of the MOEA. If $qual(A_*) \in (15, 85]$ we consider the algorithm to be in the middle-stage of convergence, while $qual(A_*) > 85$ is associated with a late-stage of convergence.

Consider we wish to solve our MOOP with a certain MOEA. Let: *a)* $p = \overline{1,100}$ be an integer value; *b)* $C_{GEN}(p)$ be the minimal number of individuals that must be computed when using GEN-MSPS in order to reach a solution set S_1 with the property that $qual(S_1) \geq p$; *c)* $C_{SSA}(p)$ be the minimal number of individuals that must be computed when using SSA-MSPS in order to reach a solution set S_2 with the property that $qual(S_2) \geq p$. For our MOOP-MOEA combination, we define the *SSA qualitative deficit* at the true hypervolume percentile p as:

$$\Delta_{qual}(p) = \left(\frac{C_{SSA}(p)}{C_{GEN}(p)} - 1 \right) \times 100 \tag{4}$$

This $\mathscr{H}$-derived measure shows the difference in the minimum number of individuals that must be computed when using the two parallelization schemes in order to reach a solution set S_p with the property that $qual(S_p) \geq p$.

In each series of tests that we have performed, we made 50 runs for each parameter configuration and we always report over averaged results. The authors are aware that a balanced comparison of non-dominated solution sets often takes into consideration more than a single performance metric [16], but we consider that, on its own, the hypervolume metric is sufficient for expressing major qualitative differences between solutions sets, especially when reporting over averaged results.

4.2 Basic Qualitative Performance Tests

In the first series of performed tests, using a constant fitness distribution (i.e. $c_v = 0$), we measured the quality of the MOEA archive (in terms of hyper-volume) after the completion of each generation (i.e. batch of 100 individuals in the case of SSA-MSPS). These results are presented in the subplots in Figure 4 marked with (a). We consider that a more useful perspective for presenting the same comparative convergence behavior information can be constructed by plotting the Δ_{qual} values of the MOEA archive (subplots marked with (b) from Figure 4).

The results of these initial tests confirm some of the findings in [6], in the sense that the GEN-MSPS is able to achieve a higher quality Pareto front than SSA-MSPS after the same number of evolved individuals in the early and middle stages of convergence for all the considered problems. Furthermore, when abstracting the behavioral shifts

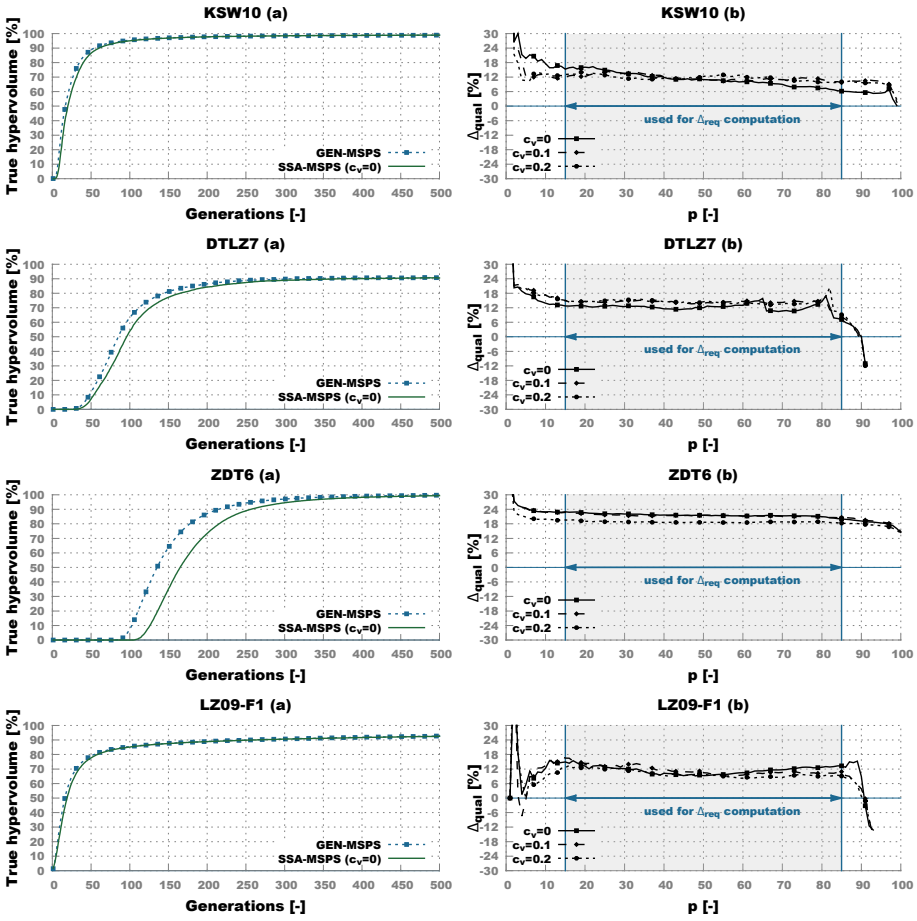

Fig. 4. SPEA2 generation-wise qualitative performance plots averaged over 50 runs

that characterize the early and late stages of convergence, we notice that Δ_{qual} values are quite constant for each test problem. By averaging the individual Δ_{qual} values associated with the middle stage of convergence (i.e., $p = \overline{16,85}$), we obtain the **average required SSA improvement** for each MOOP-MOEA combination:

$$\Delta_{req} = \frac{1}{70} * \sum_{p=16}^{85} \Delta_{qual}(p) \tag{5}$$

4.3 The Effect of Variance on the Qualitative Performance

The second series of tests that we have performed in order to gain more insight into the qualitative performance of the two parallelization schemes is again related to the influence of having variance in the time-wise fitness distribution function.

The results obtained using SPEA2 are presented in the subplots marked with (b) from Figure 4. The values of Δ_{req} for both SPEA2 and NSGA-II are shown in Table 2. It is easy to observe that, in the case of the qualitative performance, variance in the time-wise distribution of the fitness evaluation function has a negligible effect: Δ_{req} is not directly proportional to the amount of variance and, in half of the cases, the observed average changes induced on Δ_{req} by having some level of variance are not statistically significant. This creates a stark contrast when comparing with the effect that variance has on the quantitative performance (Section 3.2) and provides a solid indicator that SSA-MSPS should be favored in the presence of significant variance.

Table 2. Averaged values of the Δ_{req} metric over 50 runs for different levels of variance in the time-wise distribution of the fitness evaluation function. For each MOEA - MOOP combination, the highest value is highlighted and marked with "$^+$" if the difference between it and the lowest Δ_{req} value of the combination is statistically significant (one-sided Mann-Whitney-Wilcoxon test with a considered significance level of 0.05).

	Δ_{req} for **SPEA2** [%]				Δ_{req} for **NSGA-II** [%]			
Problem	$c_v = 0$	$c_v = 0.02$	$c_v = 0.10$	$c_v = 0.20$	$c_v = 0$	$c_v = 0.02$	$c_v = 0.10$	$c_v = 0.20$
KSW10	11.06	11.59	11.77	11.64	12.65	11.41	12.25	12.62
DTLZ7	12.13	16.16^+	14.10	14.21	18.46	20.43^+	16.58	17.79
ZDT6	21.55^+	20.22	21.39	18.75	21.91	22.33	22.67	23.32
LZ09-F1	11.49	14.02^+	11.47	10.04	12.71	13.73	12.86	14.04

4.4 The Interplay between the Quantitative and the Qualitative Aspects

This stable behavior of the qualitative deficit exhibited by SSA-MSPS allows for a simple reasoning regarding the comparative performance of GEN-MSPS and SSA-MSPS: *if, for a given optimization setup, the quantitative improvement of SSA-MSPS (Δ_{struct}) can overcompensate the qualitative deficit of SSA-MSPS (Δ_{req}), we can say that, on average, SSA-MSPS is the better parallelization choice.* This is because when ($\Delta_{struct} > \Delta_{req}$) we have good reasons to believe that, in any middle-stage convergence time interval, SSA-MSPS can produce Pareto fronts that are better or at least as good as those that might have been obtained by using GEN-MSPS for the same time interval.

Graphically, we can combine the quantitative performance and the qualitative performance by simply plotting Δ_{req} as a constant value (please see Figure 5). The intersection between the Δ_{req} constant line and the Δ_{struct} curve is the problem specific *parallelization equilibrium point*. If the equilibrium point lies to the right of the parallelization ratio, on average, SSA-MSPS is the better option, while if the equilibrium point lies to the left of the parallelization ratio, GEN-MSPS should be preferred. Figure 5 contains two charts that show the best parallelization decisions for our four test problems, under the assumptions of a constant time-wise distribution of the fitness function and of a parallelization ratio $r = 650$. SSA-MSPS is the better parallelization choice for KSW10, DTLZ7 and LZ09-F1 when using SPEA2. In the case of NSGA-II, SSA-MSPS performs better only in the case of KSW10 and LZ09-F1.

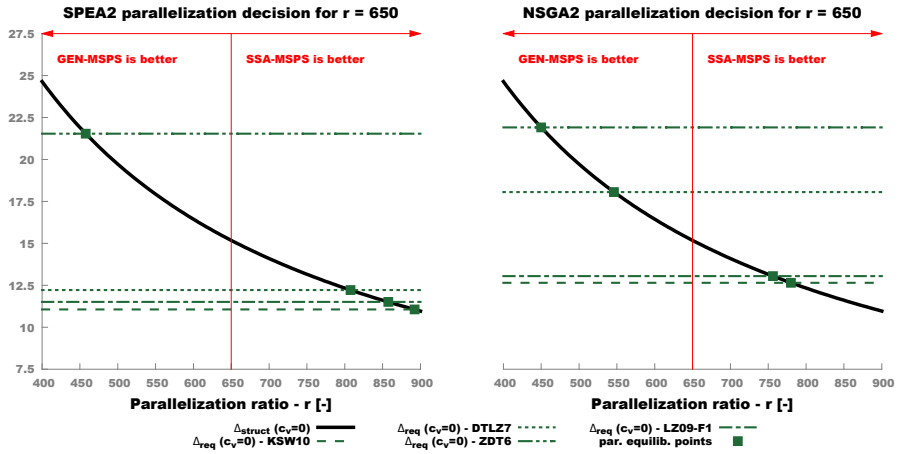

Fig. 5. Graphical example of the best parallelization decisions for specific MOOP-MOEA combinations. The hypothetical parallelization ratio is set at $r = 650$.

Applying this quantitative and qualitative analysis of parallelization performance in real world optimization scenarios is rather straightforward. Nevertheless, the posteriori character of both Δ_{struct} and Δ_{req} means that a few initial, short, mock-up test runs are required in order to estimate the concrete parallelization ratio and the problem specific qualitative behavior. This short profiling phase is more than worthwhile when one has to solve multiple MOOPs that fall within the same class (e.g., problems with roughly similar parameter vectors and / or different parameter ranges). For the same MOOP class, Δ_{req} is expected to be quite robust while Δ_{struct} can be generally estimated quite quickly as it is solely dependent on the concrete parallelization setup.

5 Conclusions and Future Work

In this paper, we investigated two master-slave parallelization methods (generational and steady state asynchronous) for MOEAs and tried to discover what are the decisive factors that can make one method outperform the other in a time-constrained optimization scenario that also requires very time-intensive fitness evaluation functions.

In order to achieve this, we performed a comparative quantitative and qualitative analysis of the behavior of these two methods when applying them to parallelize NSGA-II and SPEA2 optimization runs. The results indicate that 1) *the parallelization ratio* and especially 2) *the level of variance in the time-wise distribution of the fitness evaluation function* are the key factors that influence the relative performance of the two parallelization methods. The presence of variance is a key factor, as a rather heterogeneous fitness function can make the steady state asynchronous parallelization method (SSA-MSPS) considerably outperform its generational counterpart (GEN-MSPS).

In the future, we plan to profile using Δ_{req} more problems, including industry proposed MOOPs, and we want to test the parallelization schemes on more modern MOEAs.

References

1. Coello, C., Lamont, G., Van Veldhuisen, D.: Evolutionary Algorithms for Solving Multi-Objective Problems. Genetic and Evolutionary Computation Series, Springer (2007)
2. Deb, K.: Multi-Objective Optimization using Evolutionary Algorithms. Wiley-Interscience Series in Systems and Optimization, John Wiley & Sons, Chichester (2001)
3. Deb, K., Agrawal, R.B.: Simulated binary crossover for continuous search space. Complex Systems 9, 115–148 (1995)
4. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Transactions on Evolutionary Computation 6(2), 182 –197 (2002)
5. Deb, K., Thiele, L., Laumanns, M., Zitzler, E.: Scalable multi-objective optimization test problems. In: IEEE Congress on Evolutionary Computation (CEC 2002). pp. 825–830 (2002)
6. Durillo, J., Nebro, A., Luna, F., Alba, E.: On the effect of the steady-state selection scheme in multi-objective genetic algorithms. In: International Conference on Evolutionary Multi-Criterion Optimization (EMO 2009). pp. 183–197. Springer (2009)
7. Durillo, J.J., Nebro, A.J.: JMETAL: A Java framework for multi-objective optimization. Advances in Engineering Software 42, 760–771 (2011)
8. Durillo, J., Nebro, A., Luna, F., Alba, E.: A study of master-slave approaches to parallelize NSGA-II. In: IEEE International Symposium on Parallel and Distributed Processing (IPDPS 2008). pp. 1 –8 (2008)
9. Fleischer, M.: The measure of Pareto optima. applications to multi-objective metaheuristics. In: International Conference on Evolutionary Multi-Criterion Optimization (EMO 2003). pp. 519–533. Springer (2003)
10. Goldberg, D.E., Deb, K.: A comparative analysis of selection schemes used in genetic algorithms. In: Foundations of Genetic Algorithms. pp. 69–93. Morgan Kaufmann (1991)
11. Kursawe, F.: A variant of evolution strategies for vector optimization. In: Workshop on Parallel Problem Solving from Nature (PPSN I). Lecture Notes in Computer Science, vol. 496, pp. 193–197. Springer (1991)
12. Li, H., Zhang, Q.: Multiobjective optimization problems with complicated Pareto sets, MOEA/D and NSGA-II. IEEE Transactions on Evolutionary Computation 13(2), 284–302 (2009)
13. Yagoubi, M., Thobois, L., Schoenauert, M.: Asynchronous evolutionary multi-objective algorithms with heterogeneous evaluation costs. In: IEEE Congress on Evolutionary Computation (CEC 2011). pp. 21 –28 (2011)
14. Zitzler, E., Deb, K., Thiele, L.: Comparison of multiobjective evolutionary algorithms: Empirical results. Evolutionary computation 8(2), 173–195 (2000)
15. Zitzler, E., Laumanns, M., Thiele, L.: SPEA2: Improving the strength Pareto evolutionary algorithm for multiobjective optimization. In: Giannakoglou, K., et al. (eds.) Evolutionary Methods for Design, Optimisation and Control with Application to Industrial Problems (EUROGEN 2001). pp. 95–100. International Center for Numerical Methods in Engineering (CIMNE) (2002)
16. Zitzler, E., Thiele, L., Bader, J.: On set-based multiobjective optimization. IEEE Transactions on Evolutionary Computation 14(1), 58 –79 (2010)
17. Zăvoianu, A.C., Bramerdorfer, G., Lughofer, E., Silber, S., Amrhein, W., Klement, E.P.: A hybrid soft computing approach for optimizing design parameters of electrical drives. In: Snášel, V., et al. (eds.) International Conference on Soft Computing Models in Industrial and Environmental Applications (SOCO 2012), Advances in Intelligent Systems and Computing, vol. 188, pp. 347–358. Springer Berlin / Heidelberg (2013)