Comparing Asynchronous and Synchronous Parallelization of the SMS-EMOA

Simon Wessing[1(✉)], Günter Rudolph[1], and Dino A. Menges[2]

[1] Computer Science Department, Technische Universität Dortmund, Otto-Hahn-Str. 14, 44221 Dortmund, Germany
{simon.wessing,guenter.rudolph}@tu-dortmund.de

[2] Adept Technology GmbH, Revierstr. 5, 44379 Dortmund, Germany
dino.menges@adept.com

Abstract. We experimentally compare synchronous and asynchronous parallelization of the SMS-EMOA. We find that asynchronous parallelization usually obtains a better speed-up and is more robust to fluctuations in the evaluation time of objective functions. Simultaneously, the solution quality of both methods only degrades slightly as against the sequential variant. We even consider it possible for the parallelization to improve the quality of the solution set on some multimodal problems.

Keywords: Asynchronous · Synchronous · Parallel · Multiobjective · Evolutionary · Optimization

1 Introduction

With the rise of multi-core systems in all device classes from smartphones to desktops, parallel algorithms become more and more important. Parallelization is especially beneficial in optimization, where a high number of objective function evaluations should be enabled. An asynchronous parallelization appears preferable as it gets around the inevitable idle times caused by synchronous parallelization, but it must be precluded that this advantage is bought at the expense of solution quality.

Formerly, sophisticated algorithms containing message passing, master/slave concepts, or island models were often necessary to distribute execution on a cluster of nodes [3,5,6]. On present-day integrated multi-core architectures, simple shared memory communication may already be sufficient, especially for the application area of population-based optimization. Here, we focus on an evolutionary algorithm (EA) for multiobjective optimization, namely the S-metric selection evolutionary multiobjective optimization algorithm (SMS-EMOA) [1]. In its original form, the algorithm follows a steady-state scheme, which means that only one offspring solution is created per generation. This approach is in a sense optimal with regard to the exploitation of information in the current population, but unsuitable to synchronous parallelization. With synchronous parallelization, we mean the creation of $\lambda > 1$ offspring per generation, evaluating

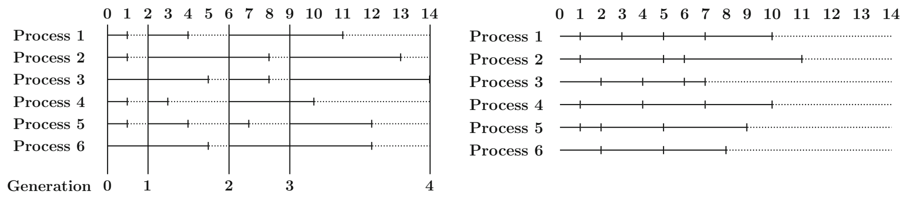


Fig. 1. Illustration of synchronous (left) and asynchronous parallelization (right). Idle times are visualized as dotted lines. In the asynchronous case, three time steps can be saved by giving up the generation concept.

them in parallel, and carrying out selection after all evaluations have finished. This approach requires an extension of the selection scheme, to be able to select μ individuals from $\mu + \lambda$. This $(\mu + \lambda)$-selection may have been avoided for SMS-EMOA to some extent, because the exact calculation of the hypervolume contributions probably requires algorithmically more complex code, or will take longer than the conventional $(\mu + 1)$-selection. Alternatively, one could sidestep the problem by taking a greedy approach of λ selections in the $(\mu + 1)$-scheme, removing one individual after another. However, Bringmann and Friedrich advise against this approach [2].

Unfortunately, the synchronization is unfavorable in case of fluctuating evaluation times, because it creates idle times (see Fig. 1). An alternative is asynchronous parallelization, which allows us to keep using $(\mu + 1)$-selection, but means giving up the concept of generations. Klinkenberg et al. [5] were the first to propose asynchronous parallelization of the SMS-EMOA. Their implementation was a master/slave approach using message passing on a cluster with twelve nodes. They combined it with metamodeling to save expensive function evaluations and applied it to a molecular control problem. The results indicated a nearly linear speed-up and only a slight decrease of solution quality between one and twelve processors. However, the analysis was not taken further because the parallelization was not the sole topic of the work. Another successful real-world application of the asynchronous SMS-EMOA is due to Menges et al. [7], who optimized the motion planning of a mobile robot.

Depolli et al. [3] investigate the asynchronous parallelization of a multiobjective differential evolution algorithm (AMS-DEMO) on a steel casting problem and benchmarks. They identify the *selection lag* as an important measure for the performance of asynchronous EMOAs. It is defined as "the number of solutions that undergo selection in the time between the observed solution's creation and selection" [3]. In their experiments, a linear speed-up with four processors could already be observed for evaluation times of only 0.01 seconds. While these results are encouraging, the question arises if the SMS-EMOA with its relatively expensive survivor selection achieves similar values. We try to answer this question with our experiment in Sect. 3. Before, we present the asynchronous SMS-EMOA in detail (Sect. 2), and afterwards we draw conclusions in Sect. 4.

Algorithm 1. Asynchronous SMS-EMOA

```
Input: $mutex$, population $P_0$
 1: $t \leftarrow 0$
 2: while stopping criterion not fulfilled do
 3:     $\boldsymbol{x} \leftarrow$ createOffspring($P_t$, $mutex$)        // create 1 offspring
 4:     evaluate($\boldsymbol{x}$)        // calculate objective values
 5:     enter($mutex$)        // lock out other processes
 6:     $Q_t \leftarrow P_t \cup \{\boldsymbol{x}\}$
 7:     $\{F_1, \dots, F_w\} \leftarrow$ nondominatedSort($Q_t$)        // sort in $w$ fronts
 8:     $\boldsymbol{r} \leftarrow$ createReferencePoint($F_w$)        // calculate reference point for last front
 9:     $\boldsymbol{x}^* \leftarrow \mathrm{argmin}_{\boldsymbol{x} \in F_w}(\Delta_s(\boldsymbol{x}, F_w, \boldsymbol{r}))$        // determine $\boldsymbol{x}^*$ with smallest contribution
10:     $P_{t+1} \leftarrow Q_t \setminus \{\boldsymbol{x}^*\}$        // remove worst individual
11:     $t \leftarrow t + 1$
12:     leave($mutex$)        // release lock for other processes
13: end while
```

2 The Asynchronous SMS-EMOA

Large differences in the evaluation times of individuals result in idle times in synchronously parallelized algorithms, because selection cannot start before all objective values are available. These idle times are of course a waste of resources. Figure 1 (left) shows an example of such a generational approach with a $(\mu+6)$-EA. The six offspring are evaluated in parallel. The time during which a process calculates a function value is indicated by a horizontal solid line. Dotted lines mark idle times. The vertical lines mark the synchronization points between the generations. In this example, 29 of the 84 time steps are unused, so the system is idle 35 % of the time. The asynchronous approach is more efficient, finishing the execution three time steps earlier and idling only 11 time steps at the end (Fig. 1, right).

Thus, we implement the SMS-EMOA as an asynchronous algorithm to minimize the idle time. The pseudocode is shown in Algorithm 1. The idea is to have several processes working on a shared population, with the additional benefit that the selection scheme can stay a $(\mu + 1)$. To make this work, all read and write operations involving the population must be protected by a lock, allowing only one process to access at a time. Entering and leaving the critical section is illustrated in lines 5 and 12. As the required functionality is provided by virtually all modern programming languages, these modifications are extremely simple. The objective function evaluation, where the most time is spent according to the common black-box optimization assumptions, may happen in parallel. After the process finishes an evaluation, it waits until it can enter the critical section to carry out the survivor selection. Naturally, the new individual may either replace another one or be rejected. Then, the section is left and the next generation starts with creating the next individual by variation. This loop continues until a stopping criterion is fulfilled. So, every single process executes all tasks of an SMS-EMOA, not only a subset as in a master/slave scenario.

Table 1. Experimental factors

Factor	Type	Symbol	Levels
Problem instances	Non-observable		{WFG1, ..., WFG9}
#Objective functions	Observable	m	$\{2, 3\}$
Evaluation base time	Observable	t_b	$\{0.01, 0.1, 1\}$
Evaluation time behavior	Observable		{fixed, random, proportional}
Parallelization	Control		{sync, async}
Parallelization degree	Control	p	$\{1, 4, 16, 64\}$
Population size	Control	μ	$\{10, 100\}$

Also the function createOffspring does a read access to the population for parent selection. It has to protect it by a critical section to avoid modifications of the population during this time.

3 Experiment

Research Question. How do the properties of multiobjective problems and parallelization settings of the SMS-EMOA influence performance?

Pre-experimental Planning. When the overhead of the SMS-EMOA is negligible in comparison to the evaluation time, an almost linear speed-up can be expected for the asynchronous variant [3,5]. The synchronous variant should only achieve the same run time with constant evaluation times, and is expected to suffer from fluctuations, as explained in Sect. 2. The hypervolume calculations of the SMS-EMOA become a bottleneck with increasing number of objectives m and increasing population size μ. Thus, it is expected that the speed-up deteriorates when m and μ are large and evaluation time is low. The experimental setup is chosen to enable quantification of this behavior.

Selection lag is recorded for both synchronous and asynchronous variants. Depolli et al. [3] identify the selection lag of the asynchronous variant (without queues) as $p - 1$ for p processors. We presume that in the synchronous case, a value of $\frac{1}{p}\sum_{i=0}^{p-1} i = (p-1)/2$ would be the expected value for the selection lag, because only individuals in the same generation can be selected during one individual's evaluation.

Task. We calculate the dominated hypervolume and the averaged Hausdorff distance (AHD, [8]) of the final population with respect to a Pareto-optimal reference set, after running the SMS-EMOA for a fixed number of function evaluations. The reference set contains 500 points for two objectives and 1000 for three. The reference points for the hypervolume are $(3, 5)^\top$ and $(3, 5, 7)^\top$. To assess running time, wall-clock time is measured and the weak speed-up in comparison to the sequential variant is computed [3]. The term *weak* speed-up means that we divide the sequential time by the parallel time without taking into account the potential quality differences of the results, which are regarded separately.

Setup. We implemented the algorithm described in Sect. 2 and the synchronous variant in the language Python (version 3.4). The code is publicly available in the packages evoalgos[1] and optproblems [9,10]. As variation operators, simulated binary crossover and polynomial mutation are used. The parameters of these operators are set to $\eta_m = 20$, $\eta_c = 20$, $p_m = 0.1$, and $p_c = 0.7$.

Table 1 contains the experimental factors for this experiment, which are combined in a full-factorial design. We carry out five stochastic replications per configuration. As test problems, the set from the walking fish group (WFG, [4]) is used with two and three objectives. The number of decision variables is set to 24, with $k = 4$ position-related parameters. The feasible region of the problems is normalized to the unit hypercube. The evaluation time of the objective functions is determined as follows. We assume a base value of t_b seconds and define three different ways to obtain the actual evaluation time t_e. The first alternative is to use the t_b value as it is, $t_e = t_b$. The second variant takes two random uniform numbers $u_1, u_2 \sim \mathcal{U}[0,1]$ and sets $t_e = (u_1 + u_2) \cdot t_b$. This way, t_e has a triangular distribution between zero and $2t_b$. The last approach uses $t_e = t_b \cdot f_1(\boldsymbol{x})$, where f_1 is the first objective function of the WFG problems, whose image is always $[0, 2]$. This setup is motivated by different real-world scenarios. The fixed evaluation time corresponds to homogeneous hardware and constant simulation time. Random fluctuations will appear if the optimization is running on a heterogeneous cluster. Simulation time may also be solution-dependent, leading to correlated evaluation times, as in [7].

The number of function evaluations is set to 10000 for each algorithm run. We exclude the population initialization and only measure the time spent in the optimization loop. By using *sleep* system calls to spend the t_e seconds, we can simulate a parallelized SMS-EMOA run on a single core, because the other algorithm parts are in critical sections anyway. The experiment is run on AMD Opteron 6276 processors with 2.3 GHz; operating system is Ubuntu Linux.

Results and Observations. Figs. 2 and 3 show the weak speed-up. The solid and dashed lines depict median values of the 45 runs on the nine WFG problems. Error bars mark 95 % confidence intervals for the median. The grey diagonal represents the maximally possible linear speed-up. In most cases, the asynchronous variant obtains a better speed-up than the synchronous one, with a few exceptions for high parallelization degrees and low evaluation times on two objectives. The speed-up of the synchronous variant sometimes even drops below one for three objectives. Figure 4 illustrates the selection lag values for the different configurations. For this figure, we first calculated the mean selection lag for each run. The lines in the figure are the median of 135 runs, due to the number of remaining configurations per panel. Generally, the predicted values seem to be accurate, except that the selection lag of the asynchronous variant drops off when evaluation times are small compared to selection times. Figure 5 shows some selected indicator values for hypervolume and averaged Hausdorff distance. The random noise in these values is much higher than for the run times. However, in Fig. 5a

[1] With a runnable example in the documentation at https://ls11-www.cs.tu-dortmund.de/people/swessing/evoalgos/doc/algo.html.

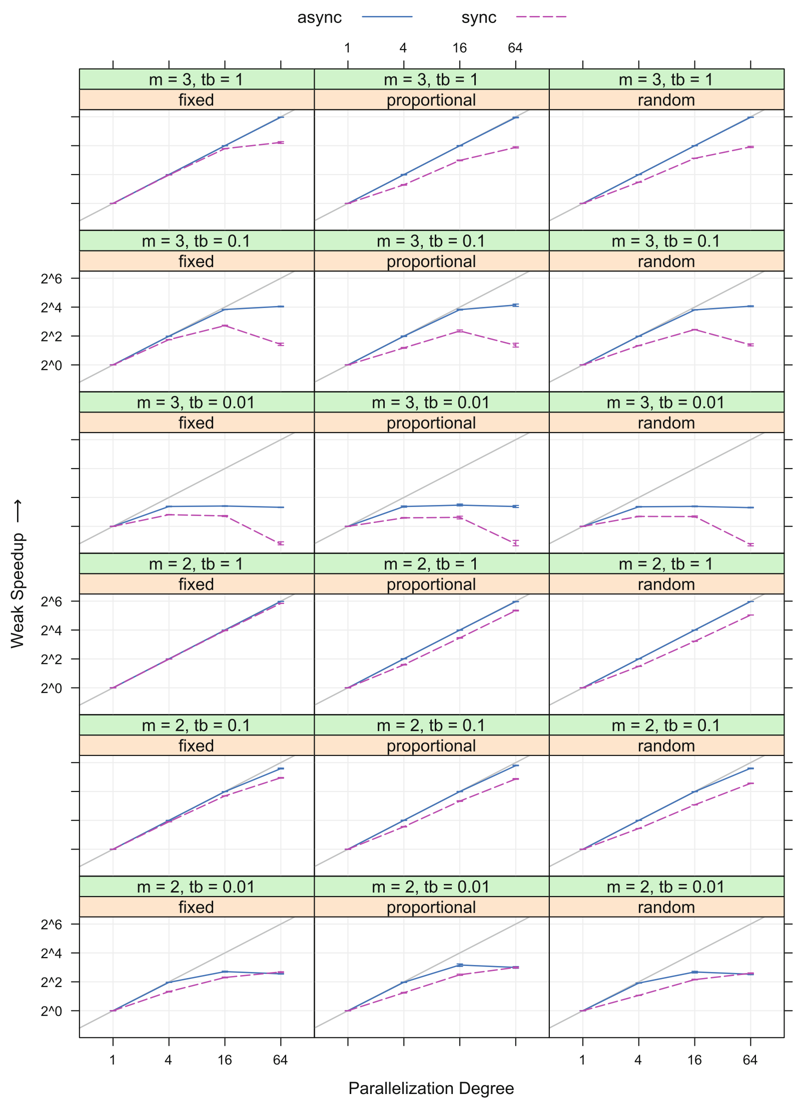


Fig. 2. Weak speed-up versus parallelization degree for $\mu = 10$.

there seems to be a positive effect on both indicators, while in Fig. 5b and c the results seem mixed.

Discussion. The cases where the synchronous beats the asynchronous variant regarding speed-up may be caused by differences in the overhead of the implementations. The asynchronous one especially makes more function calls, because each solution is processed individually. On the other hand, the decline of the synchronous variant for $m = 3$ may be because it has to compute hypervolume for

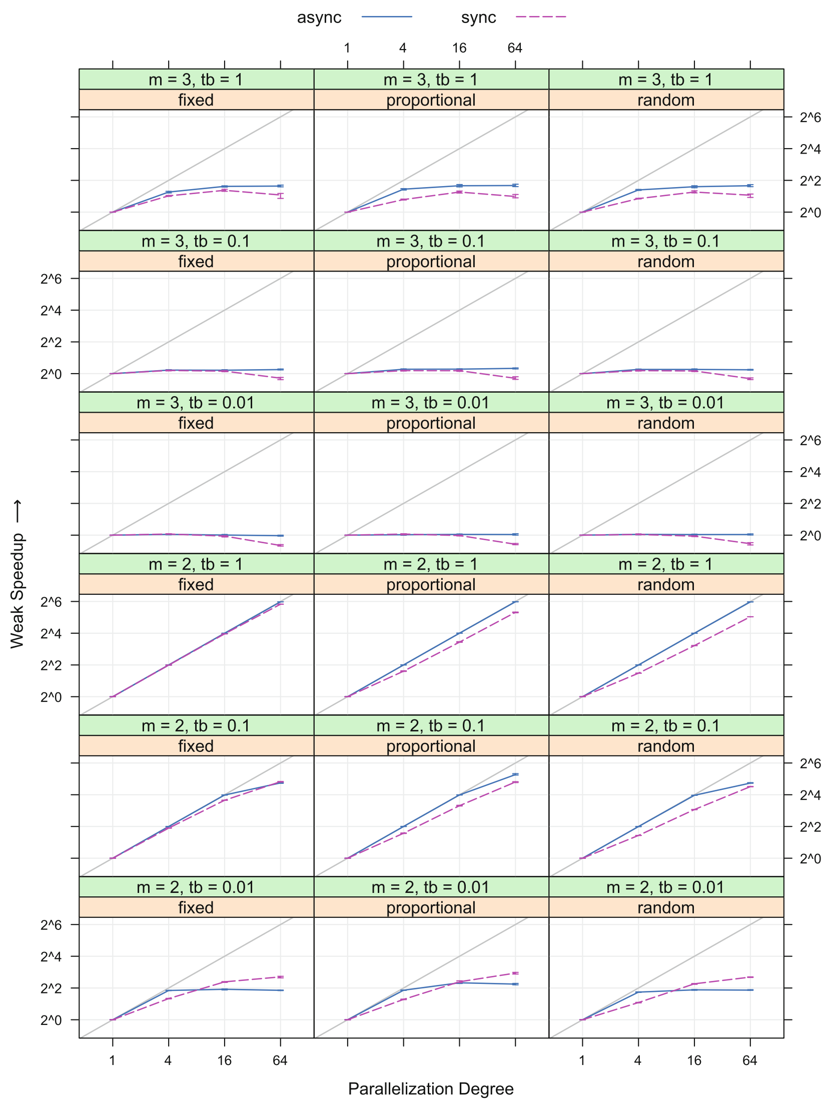


Fig. 3. Weak speed-up versus parallelization degree for $\mu = 100$.

$\mu + p$ solutions, while the asynchronous only ever does $\mu + 1$. Generally, the measured speed-ups should be seen as rather conservative estimates, if we consider that the SMS-EMOA, including the hypervolume computation, was implemented in pure Python. The speed-up could be further improved by implementing it in a lower-level language such as C++.

We are not entirely sure why the selection lag of the asynchronous variant is sometimes lower than expected. One explanation could be that the distribution of the budget on the worker processes becomes uneven with decreasing t_b. This

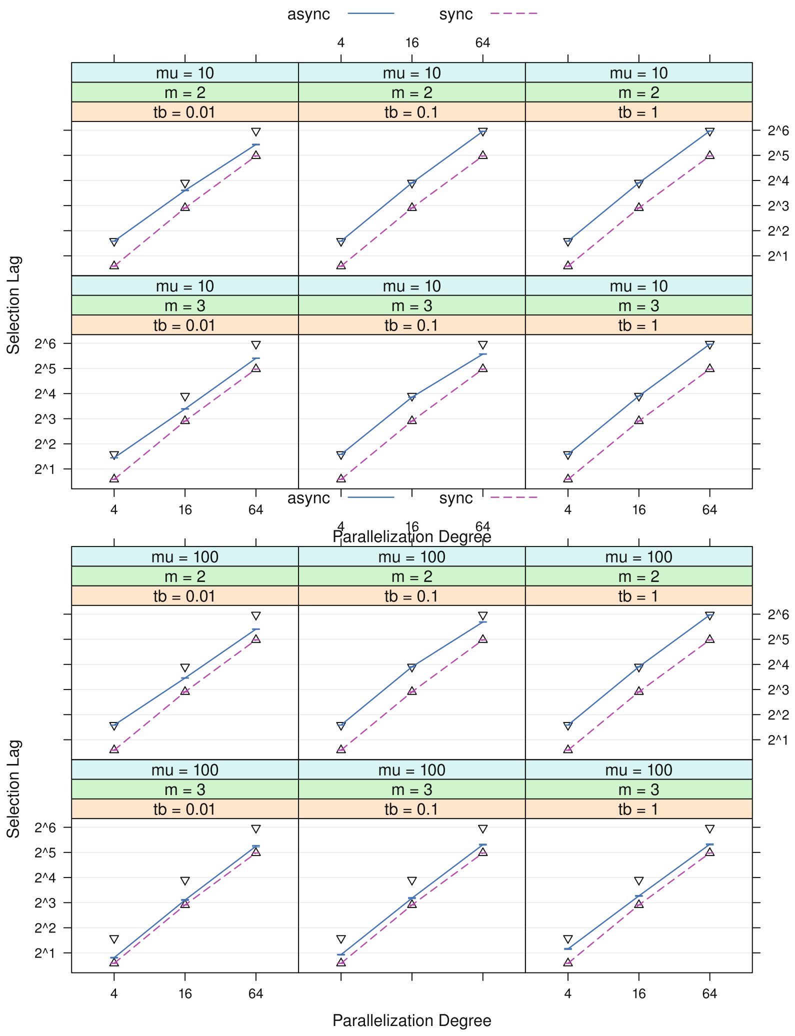
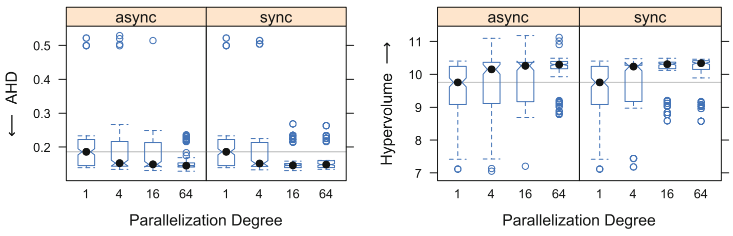


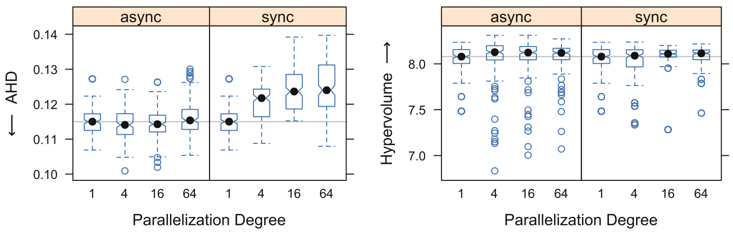
Fig. 4. Measured selection lag versus parallelization degree. The upper triangles (∇) mark $p-1$, the lower ones ($\triangle$) $(p-1)/2$.

hypothesis stems from the observation that the scheduler only ran a single thread when we switched off the delay completely. If it is true, then the selection lag could also drop below $(p-1)/2$. To look into this issue, we recorded the partition of the budget and calculated its standard deviation. However, the effect could only be observed for fixed evaluation times, and the standard deviation did not exceed the values of the two fluctuating cases. In any case, we recommend to record this data on the actual parallelization also in future experiments.

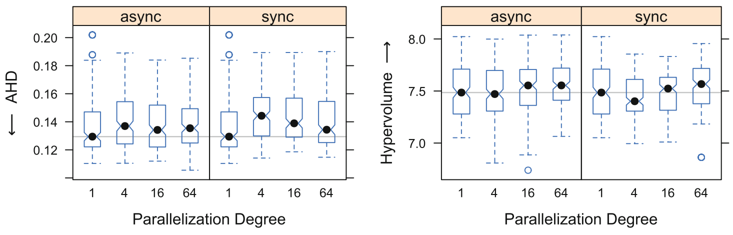
The results in Fig. 5 can be explained by the fact that for multimodal problems it is usually beneficial to spend a larger part of the budget on exploration



(a) WFG2



(b) WFG4



(c) WFG9

Fig. 5. AHD and hypervolume values of the final populations for configurations with $m = 2$ and $\mu = 10$. All three test problems are multimodal.

than for unimodal problems. The asynchronous variant additionally can sample objective space regions with lower evaluation times with a higher density than expensive regions, due to the missing synchronization. Thus, proportional evaluation times may cause a drift of the population towards a small part of the Pareto-front. This may either be beneficial, if simulation time is to be minimized [7], or detrimental, if we want to simulate interesting solutions with higher fidelity. Naturally, the assessments of AHD and hypervolume do not necessarily have to agree.

4 Conclusion

Both the synchronous variant with greedy selection and the asynchronous variant obtain an almost linear speed-up in a scenario of expensive function evaluations and moderate parallelization. The synchronous variant falls off more sharply under less favorable circumstances. The experiments have shown that the quality of solutions may even increase with more parallelism for multimodal problems whereas it may decrease for unimodal problems. We conjecture that this behavior is caused by a higher selection lag preventing a rapid movement to local optima. In theory, selection lag should only depend on the parallelization degree (and the queue size, if queues are used). However, our experiment discovered that for asynchronous parallelization, reality can somewhat deviate from theory. It is our impression that the measured selection lag especially deviates from the expectation in the cases where speed-up is low and parallelization does not work well. Thus, it might be used as a tool to assess the usefulness of asynchronous parallelization even when speed-up cannot be computed due to missing data on sequential performance.

References

1. Beume, N., Naujoks, B., Emmerich, M.: SMS-EMOA: multiobjective selection based on dominated hypervolume. Eur. J. Oper. Res. **181**(3), 1653–1669 (2007)
2. Bringmann, K., Friedrich, T.: Don't be greedy when calculating hypervolume contributions. In: Proceedings of the Tenth ACM SIGEVO Workshop on Foundations of Genetic Algorithms, FOGA 2009, pp. 103–112. ACM (2009)
3. Depolli, M., Trobec, R., Filipič, B.: Asynchronous master-slave parallelization of differential evolution for multi-objective optimization. Evol. Comput. **21**(2), 261–291 (2012)
4. Huband, S., Hingston, P., Barone, L., While, L.: A review of multiobjective test problems and a scalable test problem toolkit. IEEE Trans. Evol. Comput. **10**(5), 477–506 (2006)
5. Klinkenberg, J.W., Emmerich, M.T.M., Deutz, A.H., Shir, O.M., Bäck, T.: A reduced-cost SMS-EMOA using kriging, self-adaptation, and parallelization. In: Ehrgott, M., Naujoks, B., Stewart, J.T., Wallenius, J. (eds.) Multiple Criteria Decision Making for Sustainable Energy and Transportation Systems. Lecture Notes in Economics and Mathematical Systems, vol. 634, pp. 301–311. Springer, Heidelberg (2010)
6. Märtens, M., Izzo, D.: The asynchronous island model and NSGA-II: study of a new migration operator and its performance. In: Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation, GECCO 2013, pp. 1173–1180. ACM (2013)
7. Menges, D.A., Wessing, S., Rudolph, G.: Asynchrone Parallelisierung des SMS-EMOA zur Parameteroptimierung von mobilen Robotern. In: Hoffmann, F., Hüllermeier, E. (eds.) Proceedings 25, Workshop Computational Intelligence. Schriftenreihe des Instituts für Angewandte Informatik/Automatisierungstechnik, vol. 54, pp. 47–65. KIT Scientific Publishing (2015). (in German)
8. Schütze, O., Esquivel, X., Lara, A., Coello Coello, C.A.: Using the averaged Hausdorff distance as a performance measure in evolutionary multiobjective optimization. IEEE Trans. Evol. Comput. **16**(4), 504–522 (2012)
9. Wessing, S.: evoalgos: modular evolutionary algorithms (2016). Python package version 0.3. https://pypi.python.org/pypi/evoalgos
10. Wessing, S.: optproblems: infrastructure to define optimization problems and some test problems for black-box optimization (2016). Python package version 0.8. https://pypi.python.org/pypi/optproblems