2013 IEEE 27th International Symposium on Parallel & Distributed Processing Workshops and PhD Forum

Scalability Analysis of the Asynchronous, Master-Slave Borg Multiobjective Evolutionary Algorithm

David Hadka, Kamesh Madduri
Department of Computer Science and Engineering
The Pennsylvania State University
University Park, PA
dmh309@psu.edu, madduri@cse.psu.edu

Patrick Reed
Department of Civil Engineering
The Pennsylvania State University
University Park, PA
preed@engr.psu.edu

***Abstract*—The Borg Multiobjective Evolutionary Algorithm (MOEA) is a new, efficient, and robust optimizer that outperforms competing optimization methods on numerous complex engineering problems. To date, the Borg MOEA has been successfully applied to problems ranging from aerospace applications to water resources engineering. Problems from these domains often involve expensive design evaluations that require large-scale parallel algorithms to produce results in a reasonable amount of time. This study presents the first theoretical and experimental look at parallelizing the Borg MOEA. First, we derive theoretical models for predicting speedup, efficiency, and processor count lower and upper bounds. Second, we validate these models on a simple problem, DTLZ2, and a harder, non-separable problem, UF11. Third, we examine the effects of scaling on convergence speed and solution quality. These experiments are performed on the 62,976 core Texas Advanced Computing Center (TACC) Ranger system.**

***Keywords*-scalability, asynchronous, multiobjective, evolutionary, algorithm**

I. Introduction

Evolutionary algorithms (EAs) are efficient search heuristics that are being applied to successfully find acceptable solutions to challenging problems in various science and engineering disciplines [1], [2], [3], [4]. Problems from these disciplines are often complex and time-consuming, and require parallelization in order to reach acceptable solutions in a reasonable amount of time. As a result, the study and application of parallel EAs is rapidly expanding.

Parallel EAs can be classified into two categories: synchronous and asynchronous [5]. Synchronous EAs require that all of their population members are evaluated in a given generation before their evolutionary search proceeds to the next generation. Synchronization poses a computational bottleneck that leads to large algorithmic overhead and strongly limits the maximum parallelization efficiencies that can be achieved. Alternatively, asynchronous EAs avoid the synchronization step and eliminate their dependence on fully evaluated populations (or generations), greatly expanding their potential to yield greater parallel efficiencies.

Since Bethke's [6] first attempt to parallelize a genetic algorithm in 1976, many studies have experimentally explored various methods of parallelizing EAs. For years, researchers developed different parallelization strategies and applied these algorithms successfully in many problem domains [7]. Then, in 1997, Erick Cantú-Paz began developing strong theoretical models for designing efficient parallel EAs [8], [9], [10], which lead to his seminal publication [5] detailing the theoretical properties of parallel EAs. Cantú-Paz's derivations focused on synchronous EAs, commenting only that asynchronous EAs would likely yield significant improvements in efficiency. However, to date, no detailed theoretical analysis of the scalability of asynchronous EAs has been published.

This study presents the first such detailed scalability analysis of parallel, asynchronous multi-objective EAs (MOEAs). MOEAs differ from classical EAs by optimizing a suite of conflicting objectives to approximate Pareto optimal solutions (i.e., those solutions where performance in one objective cannot be improved without degrading one or more other objectives) [11]. Relative to traditional single objective EAs, parallelizing MOEAs is challenging because of the problem's increased complexities and the algorithm's increased overhead.

In particular, we investigate the scalability of the asynchronous, master-slave Borg MOEA [12]. The Borg MOEA is notable for its ability to adapt its search operators at runtime, allowing it to rapidly adapt itself to problems with widely varying characteristics. This ability to adapt to various problem domains has been demonstrated on a number of analytical and real-world problems [13], [14], [15]. Providing a parallel implementation of the Borg MOEA is therefore paramount to maximize its use on real-world problems with complex, time-consuming objectives.

In this study, we explore the parallel scaling limits of the Borg MOEA by developing an analytical model and a simulation model. The analytical model assumes a well-characterized, constant function evaluation time (T_F). This allows us to derive closed-form solutions for the expected speedup, efficiency, and processor count lower and upper bounds. The simulation model allows us to better model communication costs, critical section overheads, and function evaluation times that follow probability distributions. Using these models, we show how we are able to more accurately model the parallel execution of the Borg MOEA.

This study provides several contributions to the fields of parallel computing and evolutionary algorithms. First, we derive theoretical models for the scalability of asynchronous, master-slave MOEAs. This includes using a simulation model to more accurately model the runtime of the algorithm than is possible with analytical models. Second, we validate the simulation model on the $62,976$ core TACC Ranger system, showing that our model provides accurate predictions of runtime, speedup and efficiency. Third, we present the first scalability results on the new, asynchronous, master-slave Borg MOEA. To this end, we show that the simulation model is able to accurately predict the scalability of the Borg MOEA, which can now be used to design the parallel topology of the Borg MOEA to maximize efficiency and solution quality. Finally, we provide the first theoretical comparison of synchronous and asynchronous MOEAs, showing that the asynchronous model is able to efficiently scale to larger processor counts.

The remainder of this paper is organized as follows. Section II introduces the Borg MOEA in detail and introduces the terminology used throughout this paper. Section III develops an analytical model for the serial Borg MOEA. Section IV constructs the analytical model and simulation model for the asynchronous, master-slave Borg MOEA. Section V overviews the experimental study where these models are validated against two problems. The results of this experiment are reported in Section VI. Lastly, the impact of this work is discussed in Section VII.

II. The Borg MOEA

The Borg MOEA is a state-of-the-art optimization algorithm first introduced in Hadka and Reed [12]. Like other evolutionary algorithms, the Borg MOEA operates by evolving a population of candidate solutions towards solutions with higher fitness. However, Borg introduces several novel features to improve its convergence speed, efficiency, and reliability.

Traditional evolutionary algorithms use a single search operator to evolve the candidate solutions in the population. Knowing which search operator will effectively solve an arbitrary problem is impossible to know in practice [16]. Instead, the Borg MOEA employs an ensemble of search operators to evolve the population. The probability of applying each operator is auto-adapted based on its ability to produce offspring with higher fitness. Operators that consistently generate highly fit offspring will be granted a higher probability of use in the future, effectively tailoring its operators to the problem at hand.

The same six real-valued operators are used in this study as have been used in prior studies [15], [13], [14]. This includes simulated binary crossover (SBX) [17], differential evolution (DE) [18], parent-centric crossover (PCX) [19], simplex crossover (SPX) [20], unimodal normal distribution crossover (UNDX) [21], and uniform mutation (UM) applied with probability $1/L$.

The Borg MOEA uses an ϵ-dominance archive to track the Pareto-dominant solutions [22]. Since the ϵ-dominance archive maintains the best solutions in terms of diversity and convergence produced by the algorithm, the Borg MOEA favors operators that contribute solutions to the ϵ-dominance archive. Thus, operators that consistently contribute high-quality and diverse solutions are given a higher probability of use in future iterations.

The Borg MOEA also features several mechanisms to prevent preconvergence. Preconvergence occurs when an evolutionary algorithm finds and converges to a local optima, and as a result is unable to discover the true global optimum [23], [24]. The Borg MOEA monitors the ϵ-dominance archive for preconvergence by monitoring changes to the ϵ-dominance archive. If the archive remains unchanged for a period of time, then search has stagnated and preconverged.

If preconvergence is detected, the Borg MOEA triggers a restart. The goal of a restart is to increase the genetic diversity in the population in an attempt to escape the local optima and discover the global optimum. During a restart, the population is first emptied and filled with the Pareto-dominant solutions stored in the archive. Next, diversity is introduced by taking members of the archive, applying uniform mutation with probability $1/L$, and adding the mutated solutions to the population. The Borg MOEA may also choose to adapt the population size and selection pressure at this time. It has been shown that maintaining a population size proportional to the problem difficulty is essential to preventing preconvergence [25].

The Borg MOEA has been demonstrated successfully on a number of real-world engineering problems [15], [14]. In these studies, it has consistently and sometimes substantially outperformed other high-profile optimization algorithms. For instance, in Hadka et al. [14], the Borg MOEA is used to design three general aviation aircraft while satisfying 9 economic and performance constraints. While other high-profile optimization algorithms like MOEA/D [26] struggled to even find feasible solutions, the Borg MOEA not only quickly found feasible solutions but produced aircraft designs that outperformed the best-known results to date. For this reason, we have parallelized the Borg MOEA to enable its use on large, complex engineering problems with expensive function evaluations.

The parallel, master-slave implementation of the Borg MOEA follows a similar structure as the serial Borg MOEA. The execution of the serial Borg MOEA consists of the following ordered steps: 1) apply the search operator to produce an offspring; 2) evaluate the offspring; 3) add the offspring to the population and archive; and 4) periodically check for stagnation (triggering a restart if necessary) and

Table I
THE NOTATION USED THROUGHOUT THIS PAPER.

Symbol	Description
T_F	Function evaluation time
T_C	Communication time
T_A	Algorithm overhead (population management, offspring generation, etc.)
T_S	Runtime of the serial algorithm
T_P	Runtime of the parallel, asynchronous algorithm
N	Number of function evaluations
P	Number of processors

update the operator probabilities. The asynchronous, master-slave implementation works by distributing the evaluation of offspring to many worker nodes. Instead of generating and evaluating one offspring at a time, the master-slave implementation generates a new offspring whenever a worker node is available.

For analysis, we model the time required to perform each of these execution steps using the notations T_C, T_F, and T_A. T_C is the time required to send and receive messages between the master and worker nodes. T_F is the time to evaluate one offspring. T_A is the time required to perform the serial components of the Borg MOEA, including adding the offspring to the population and archive, checking for stagnation, performing restarts, and adapting the operator probabilities. The total number of function evaluations allocated to the MOEA in a single run is denoted by N, and the number of processors available to the parallel algorithm is denoted by P. When P processors are available, one acts as a master node and $P - 1$ serve as worker nodes. This notation is summarized in Table I.

Before beginning the scalability analysis, it is important to distinguish the difference between synchronous and asynchronous MOEAs. Most MOEAs in use today are generational, meaning that the population is evolved in distinct stages called generations. In a single generation, the population is evolved to produce offspring, the offspring are evaluated, and the offspring are added back into the population (possibly replacing existing members in the population). The key point here is that the previous generation must complete before the next generation starts. The need to synchronize generations gives rise to the term synchronous MOEA.

Figure 1 shows the timeline of events for the synchronous, master-slave MOEA. The vertical dotted lines indicate the start of a new generation. A generation begins when the master generates the offspring and sends them to the worker nodes for evaluation. Note that the master is also responsible for evaluating one offspring. It is also possible to send multiple solutions to a single worker node. In this study, however, we consider only the case where a single solution is sent. Next, the workers evaluate the offspring and send the results back to the master node. Finally, the master updates the population with the offspring.

Asynchronous MOEAs eliminate the concept of a generation. As soon as an offspring is evaluated and returned to the master, the next offspring is immediately generated. This is shown in Figure 2. Note that as soon as the master receives the evaluated results from a worker, it immediately produces another offspring for that worker. Also note that the T_A for the asynchronous MOEA is shorter than the T_A for the synchronous MOEA. This is because the asynchronous MOEA processes one offspring at a time, whereas the synchronous MOEA processes all offspring from the generation at once. Comparing Figures 1 and 2, one can immediately see the reduction in idle time using an asynchronous MOEA. In the remainder of this paper, we will develop and validate theoretical models for the asynchronous MOEA.

III. SERIAL MODEL

In order to calculate speedup and efficiency, we need to derive the time for the serial algorithm. The total time for running a steady-state MOEA like the Borg MOEA in serial, denoted by T_S, is shown below.

$$T_S = N\left(T_F + T_A\right) \tag{1}$$

The serial algorithm requires N function evaluations, where each function evaluation requires T_F and T_A, the time to generate the next offspring, the time to evaluate the offspring, and the time to process the evaluated offspring.

IV. ASYNCHRONOUS MODELS

In this section, we derive scalability models for asynchronous, master-slave MOEAs. Because the asynchronous model does not include any synchronization barriers, the worker nodes must compete with one another for access to the limited resources of the master node. We start by building a simple, analytical model and work our way to a more accurate but complex simulation model.

A. Analytical Model

We begin our analysis by assuming that T_F, T_A, and T_C are constant. Assuming these times are constant allows us to model the asynchronous algorithm using an analytical model. Furthermore, by assuming all communication times are constant, all steps are performed in lockstep. The master node is guaranteed to be available when a worker node completes evaluating the objectives, eliminating any resource contention. This is seen clearly in Figure 2, where the master

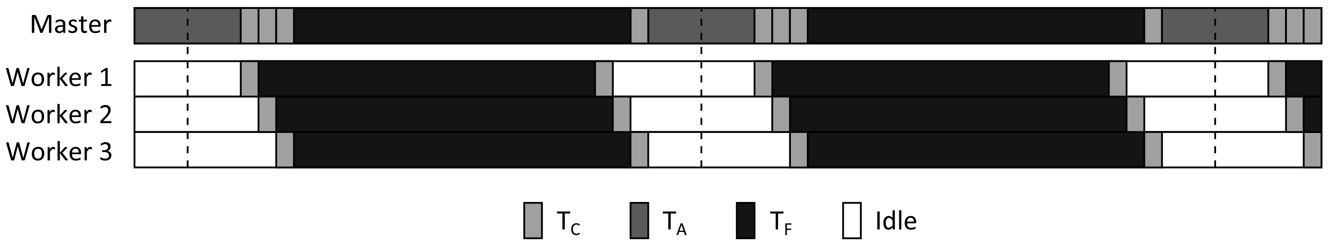


Figure 1. Diagram depicting the various costs incurred during a run of a synchronous, master-slave MOEA. In this example, $P = 4$ with one master and 3 workers. The dotted line indicates the start of a new generation.

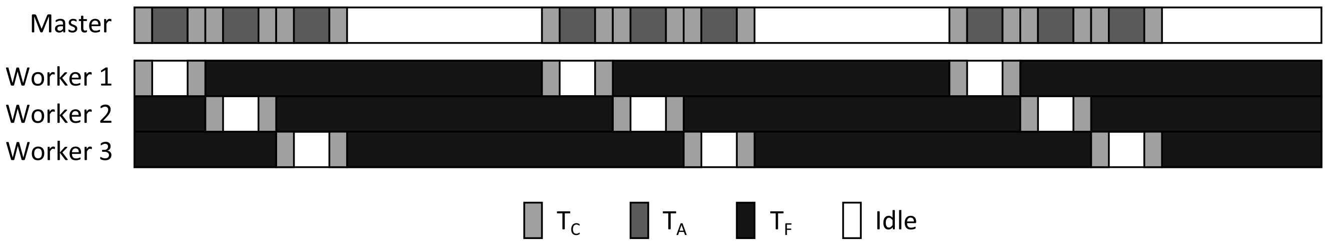


Figure 2. Diagram depicting the various costs incurred during a run of an asynchronous, master-slave MOEA. In this example, $P = 4$ with one master and 3 workers. The master sends a solution to an available worker (T_C), the worker evaluates the solution (T_F), the worker sends the evaluated solution back to the master (T_C), and the master processes the solution and generates the next offspring to evaluate (T_A).

node is always free to receive the result from a worker node as soon as the worker finishes evaluating the solution. From Figure 2, the parallel runtime of an asynchronous, master-slave MOEA is given by:

$$T_P = \frac{N}{P-1}\left(T_F + 2T_C + T_A\right) \tag{2}$$

One function evaluation requires $T_F + 2T_C + T_A$. This accounts for sending the decision variables to the worker node (T_C), evaluating the objectives (T_F), sending the objectives back to the master (T_C), processing the evaluated solution and generating the next offspring (T_A). As the evaluations are spread across all worker nodes, each worker node will perform $\frac{N}{P-1}$ evaluations.

One factor limiting scalability is the availability of the master node. We calculate the maximum number of processors that are feasible before the master reaches saturation (no idle time to process additional solutions) as:

$$P^{\text{UB}} = \frac{T_F}{2T_C + T_A} \tag{3}$$

Looking at this from a different perspective, we can ask how many processors are required to ensure the parallel implementation is at least faster than the serial implementation. To answer this question, we solve $\frac{T_S}{T_P} > 1$ to get:

$$P^{\text{LB}} > 2 + \frac{2T_C}{T_F + T_A} \tag{4}$$

From (4), we can calculate the lower bound on the number of processors needed for different time costs. Observe that the asynchronous model needs at least three processors to run faster than the serial algorithm regardless of the values of T_F, T_C, and T_A.

B. Simulation Model

The analytical model is limited by the assumptions that T_F, T_C, and T_A are constant. Relaxing these assumptions, we assume that T_F, T_C, and T_A follow a probability distribution. This introduces resource contention where the worker nodes must compete for access to the master node. When the master node is busy processing a request, worker nodes must wait in a queue until the master becomes available. This waiting will reduce the efficiency of the algorithm as P increases and resource contention becomes more likely. To model this more complex interaction, we build a simulation model.

The simulation model was developed in SimPy 2.3[1], a simulation library for Python. The structure of the simulation model is identical to that of the Borg MOEA. However, instead of actually performing the calculations or sending messages, the simulation model "holds" the resources for a set amount of time. For example, the master node would be modeled as follows. First, a "request" for the master simulates the worker waiting while the master is busy. Second, once the master is available, we "hold" the master to simulate the communication and algorithm processing time. Once the hold completes, the master is "released" and a worker is "activated". This release and activation is used to simulate sending a message to the worker node. This seqence of steps can be simulated in SimPy as shown below.

[1]http://simpy.sourceforge.net/

```
yield request, self, master
yield hold, self, sampleTc() + \
    sampleTa() + sampleTc()
yield release, self, master
activate(worker, worker.evaluate())
```

Accurate measurements of T_C, T_A, and T_F are needed for the simulation model to be accurate. Given a large sampling of these timing values for real executions of an algorithm on a parallel system, we used the R Project[2], an open-source language for statistical computing, to fit the sampled data to various distributions. Subsequently, the log-likelihood is calculated for each distribution to determine which best fits the sampled data.

Running the resulting simulation model, we can compute the simulated runtime of the parallel algorithm, T_P. From this, we can predict the efficiency of the parallel algorithm with $E_P = \frac{T_S}{PT_P}$.

V. Experiment Details

The experiment performed in this study was executed on the Texas Advanced Computing Center (TACC) Ranger system. TACC Ranger consists of 3,936 16-way symmetric multiprocessing (SMP) compute nodes, each containing four 2.3 GHz AMD Opteron Quad-Core 64-bit processors and 32 GBs of memory. Each core can perform 9.2 GFLOPS. In total, there are $62,976$ processing cores. Nodes are connected using two large Sun InfiniBand DataCenter switches.

The asynchronous, master-slave Borg MOEA was implemented in C and powered by OpenMPI. On TACC Ranger, we were able to collect timing data with a resolution of 1 microsecond. This timing data was subsequently used to approximate the probability distributions of T_A and T_F, as described in Section IV-B. The value for T_C was captured separately by measuring the round-trip time to send and receive messages between the master and all worker nodes. This allows an accurate estimation of the point-to-point communication cost since the payload of each message is a constant size. On TACC Ranger, we calculated the value of T_C to be 6 microseconds.

The Borg MOEA was applied to two well-known analytical problems. The first, the 5-objective DTLZ2 [27], is considered simple for MOEAs to solve. All decision variables are separable, and can be evolved independently of the others. The second problem, the 5-objective UF11 [28], is a variant of DTLZ2 where the decision variables are rotated and scaled to introduce dependencies between the variables. Results from the IEEE CEC 2009 competition [29] shows many state-of-the-art MOEAs struggle to solve UF11. These functions provide a mechanism to explore how problem difficulty interplays with speedup to impact the Borg MOEA's search.

[2]http://www.r-project.org

The function evaluation time of these two problems is less than 1 microsecond. To explore the scalability of the Borg MOEA, controlled delays were introduced to the problems. In this study, we explored three different delays: 0.001, 0.01, and 0.1 seconds with a coefficient of variation of 0.1. This allows us to precisely control T_F and vary it proportionally to T_C and T_A to measure the actual speedup and efficiency.

To reiterate, we are experimenting with two well-known optimization problems. Since they have been benchmarked thoroughly in the literature, they can be used to detect changes in the algorithm's behavior and dynamics resulting from parallelization. Additionally, we have introduced controlled time delays to the function evaluations to allow us precise control over the probability distribution of T_F. From this, we can observe the speedup and efficiency of the parallel algorithm across a range of function evaluation times. These two experimental controls allow us to carefully observe any dependencies between speedup and search quality.

For each problem and time delay, the Borg MOEA was executed on the TACC Ranger system with 16, 32, 64, 128, 256, 512, and 1024 processors. Each run of the algorithm was replicated 50 times, and the reported results are averaged across the 50 replicates.

VI. Results

Table II shows the results of the experimental study described in Section V with the predictions from the analytical and simulation model. The table shows the mean values for T_A, T_C, and T_F collected from the experiment. Using these time values, we can compute the predicted elapsed time from the analytical and simulation models. These predicted times are shown in Table II along with the relative error calculated with:

$$\text{Error} = \frac{\left|T_P^{\text{Actual}} - T_P^{\text{Predicted}}\right|}{\left|T_P^{\text{Actual}}\right|} \tag{5}$$

As hypothesized, the analytical model becomes error prone when the $\frac{T_F}{2T_C+T_A}$ ratio is small. This is seen by comparing the analytical model error as T_F increases. The error also increases as the processor count increases. This shows the fundamental limitation of the analytical model. It is unable to account for the resource contention encountered when a large number of worker nodes are attempting to communicate with the master node. This bottleneck is more accurately modeled by the simulation model, as indicated by the significantly lower error rates.

Also note in Table II the efficiency values recorded during this experiment. There is a clear peak in efficiency, where using fewer processors is underutilizing the system but using more processors increases resource contention. Consider how this compares to the processor count upper bound, which calculates the number of processors to saturate the master node. For demonstration, lets select the DTLZ2 case

Table II
TABLE COMPARING THE EXPERIMENTAL RESULTS TO THE ANALYTICAL AND SIMULATION MODELS. ALL TIMES ARE IN SECONDS. ERRORS ARE PERCENT DEVIATION FROM EXPERIMENTAL TIMES.

					Experimental Results		Analytical Model		Simulation Model	
Problem	P	T_A	T_C	T_F	Time	Efficiency	Time	Error	Time	Error
DTLZ2	16	0.000 023	0.000 006	0.001	9.2	0.69	7.1	23%	7.2	22%
	32	0.000 025	0.000 006	0.001	6.3	0.51	3.5	45%	5.6	12%
	64	0.000 027	0.000 006	0.001	5.8	0.28	1.7	71%	6.0	4%
	128	0.000 029	0.000 006	0.001	6.3	0.13	0.9	86%	6.4	2%
	256	0.000 031	0.000 006	0.001	6.9	0.06	0.5	93%	6.8	2%
	512	0.000 037	0.000 006	0.001	7.9	0.03	0.3	97%	8.0	2%
	1024	0.000 045	0.000 006	0.001	9.4	0.01	0.2	98%	9.6	3%
	16	0.000 023	0.000 006	0.01	67.5	0.93	67.1	1%	67.1	1%
	32	0.000 025	0.000 006	0.01	33.1	0.95	32.5	2%	32.5	2%
	64	0.000 027	0.000 006	0.01	16.6	0.94	16.0	4%	16.0	4%
	128	0.000 029	0.000 006	0.01	8.8	0.89	8.0	10%	8.0	10%
	256	0.000 031	0.000 006	0.01	6.9	0.57	4.0	43%	6.8	2%
	512	0.000 037	0.000 006	0.01	7.8	0.25	2.0	75%	8.0	3%
	1024	0.000 045	0.000 006	0.01	9.4	0.10	1.0	90%	9.6	3%
	16	0.000 023	0.000 006	0.1	667.8	0.94	667.1	1%	667.4	1%
	32	0.000 025	0.000 006	0.1	323.1	0.97	322.8	1%	323.0	1%
	64	0.000 027	0.000 006	0.1	159.0	0.98	158.9	1%	159.0	0%
	128	0.000 029	0.000 006	0.1	79.0	0.99	78.8	1%	78.9	1%
	256	0.000 031	0.000 006	0.1	39.5	0.99	39.3	1%	39.3	1%
	512	0.000 037	0.000 006	0.1	19.9	0.98	19.6	2%	19.7	2%
	1024	0.000 045	0.000 006	0.1	11.5	0.85	9.8	15%	10.0	14%
UF11	16	0.000 055	0.000 006	0.001	12.3	0.54	7.5	40%	11.6	6%
	32	0.000 057	0.000 006	0.001	11.2	0.29	3.7	67%	12.0	8%
	64	0.000 059	0.000 006	0.001	11.5	0.14	1.8	85%	12.4	8%
	128	0.000 061	0.000 006	0.001	11.8	0.07	0.9	93%	12.4	6%
	256	0.000 064	0.000 006	0.001	13.3	0.03	0.5	97%	13.4	1%
	512	0.000 068	0.000 006	0.001	14.2	0.01	0.3	98%	14.2	0%
	1024	0.000 078	0.000 006	0.001	16.3	0.01	0.2	99%	16.1	2%
	16	0.000 055	0.000 006	0.01	68.5	0.92	67.5	2%	67.6	2%
	32	0.000 057	0.000 006	0.01	35.2	0.89	32.7	8%	32.8	7%
	64	0.000 059	0.000 006	0.01	18.4	0.85	16.1	13%	16.3	12%
	128	0.000 061	0.000 006	0.01	12.6	0.62	8.0	37%	12.4	2%
	256	0.000 064	0.000 006	0.01	13.4	0.29	4.0	71%	13.4	0%
	512	0.000 068	0.000 006	0.01	14.2	0.14	2.0	86%	14.2	0%
	1024	0.000 078	0.000 006	0.01	16.2	0.06	1.0	94%	16.1	1%
	16	0.000 055	0.000 006	0.1	668.7	0.94	667.5	1%	667.8	1%
	32	0.000 057	0.000 006	0.1	323.4	0.97	323.0	1%	323.2	1%
	64	0.000 059	0.000 006	0.1	159.3	0.98	159.0	1%	159.0	1%
	128	0.000 061	0.000 006	0.1	79.2	0.99	78.9	1%	79.0	1%
	256	0.000 064	0.000 006	0.1	39.8	0.98	39.3	2%	39.4	2%
	512	0.000 068	0.000 006	0.1	20.8	0.94	19.6	6%	19.7	6%
	1024	0.000 078	0.000 006	0.1	16.6	0.59	9.8	41%	16.3	2%

where $T_A = 0.000029$, $T_C = 0.000006$, and $T_A = 0.01$. From (3), the processor count upper bound is 244. However, as seen in Table II, the peak efficiency occurs with approximately 32 processors. Maximizing the efficiency of the MOEA will require using fewer processors than the analytical model suggests.

This suggests that in situations where a large processor count is available and T_F is too small to run efficiently, better resource utilization may be possible with hierarchical topologies [5], [11]. Instead of running a single, large master-slave MOEA, the hierarchical topology runs several smaller, concurrently-running master-slave instances. Each of these instances runs on a distinct subset of the available processors. Our parallel performance simulation model can be used to determine the size of these subsets to maximize efficiency.

A. Speedup

Speedup measures how much faster a parallel algorithm is compared to its equivalent serial algorithm, calculated by $S_P = \frac{T_S}{T_P}$. Since MOEAs are stochastic processes, the quality of the results produced by an MOEA can be different for every run, and may vary across different processor counts. The quality of the results must be accounted for when computing speedup.

We measure the quality of the results using the hypervolume metric [30]. The hypervolume metric compares the approximation set produced by the Borg MOEA to the optimal result, called the reference set. Larger hypervolume values indicate the Borg MOEA produced better approximations of the reference set. Both of the tested problems have known reference sets, so the hypervolume values are relative to an ideal mathematical baseline. A hypervolume value of 1 is ideal.

Consider a fixed hypervolume threshold, h. Let T_S^h be the average time needed for the serial MOEA to produce results with a hypervolume meeting or exceeding h, and T_P^h the average time for the parallel MOEA to do the same. Then $S_P^h = \frac{T_S^h}{T_P^h}$ is the speedup observed while attaining a hypervolume of h. As h approaches its ideal value of 1, we can analyze how the Borg MOEA sustains hypervolume-based speedup to attain high-quality search results.

Figures 3 and 4 show the speedup needed to reach different hypervolume thresholds. The x-axis ranges from low-quality results ($h = 0.1$) to high-quality results ($h = 1.0$). The y-axis shows the speedup observed while attaining the hypervolume threshold. The line series represent the different processor counts tested in this study.

These figures provide several key insights. First, when solution quality is accounted for, very high levels of solution quality and problem difficulty enhance the value of parallelization. Second, when the predicted speedup is high based solely on time, it yields more time for evolution and constant levels of hypervolume speedup, S_P^h. In other words, when the parallel Borg MOEA is running efficiently, then it consistently maintains strong search quality. When the parallel Borg MOEA is running inefficiently (large P and small T_F), then a nonlinear relationship arises between speedup and search difficulty.

For example, observe the $T_F = 0.01$ subplot in Figure 3. When the parallel Borg MOEA is running efficiently with 16, 32, and 64 processors, the speedup lines are flat. This indicates the Borg MOEA is maintaining strong search quality while achieving significant speedup. However, going to 128 processors and beyond, the algorithm becomes inefficient (see the experimental result efficiencies in Table II). For larger hypervolume thresholds, the speedup is a nonlinear curve that is dependent on the desired search quality (the hypervolume threshold). On the UF11 problem in Figure 4, this nonlinearity is even more pronounced, indicating a dependency between speedup and problem difficulty.

B. Comparison to Synchronous MOEA

Building on our validated results for the simulation model, we now compare the scalability of the Borg MOEA to the synchronous MOEA developed by Cantú-Paz [5]. The analytical model developed by Cantú-Paz provides the following formula for the runtime of the parallel, synchronous MOEA:

$$T_P^{\text{Sync}} = \frac{N}{P}\left(T_F + PT_C + T_A^{\text{Sync}}\right) \tag{6}$$

Note that again we assume that each node processes only one solution per generation. Thus, P is both the processor count and population size. It is possible to have nodes evaluate more than one solution, potentially increasing efficiency when T_F and/or P is small. Cantú-Paz explores this in detail [5]. Also note that, in general, $T_A^{\text{Sync}} \approx PT_A$ since the synchronous algorithm has to process all P offspring at once.

Figure 5 shows the predicted efficiency from both models across a range of T_F and P values. T_F ranges from 0.0001 up to 1 second, and P ranges from 2 to $16,384$ processors. Complex engineered systems design has been strongly limited by computational barriers where evaluation times greatly exceed 1 second or more [31], so understanding scaling limits with large T_F and P is important. For both models, T_A and T_C are fixed at 0.000006 and 0.000060 seconds, respectively.

Note that the synchronous MOEA is able to achieve higher efficiency with smaller T_F and P. The asynchronous model appears to have a lower bound processor count of 16 and a lower bound T_F of 0.01 seconds. However, the asynchronous model is able to scale to larger processor counts than the synchronous model with the same T_F. As discussed, this is a result of the asynchronous model not requiring synchronization barriers at each generation. This provides the first theoretical results that explain in detail the conditions necessary for the asynchronous model to efficiently scale to larger processor counts than the commonly used synchronous model.

Another substantial difference between the synchronous and asynchronous MOEA is the impact on performance of highly-variable function evaluation times (T_F). Since the synchronous MOEA must wait for all workers to complete each generation, all nodes sit idle waiting for the longest running evaluation to complete. The asynchronous MOEA, on the other hand, is able to immediately send another offspring to a worker as soon as it finishes the previous

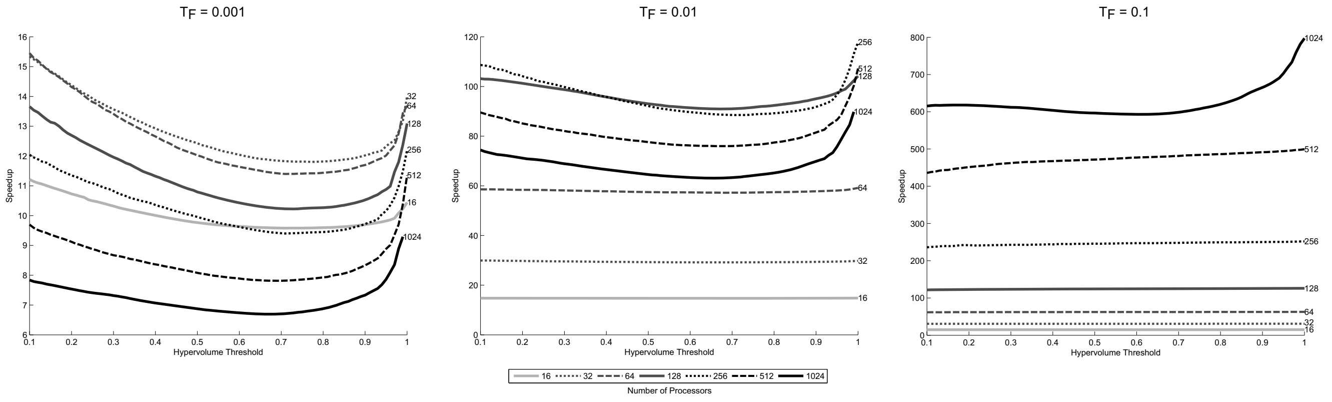


Figure 3. Parallel speedup to reach various hypervolume thresholds on the 5-objective DTLZ2 problem. Higher hypervolume thresholds correspond to higher-quality results.

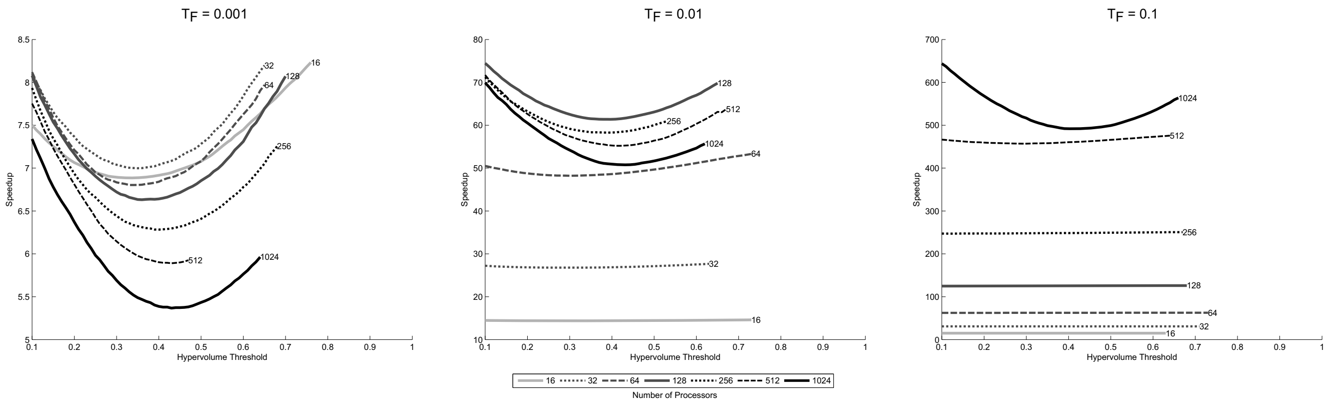
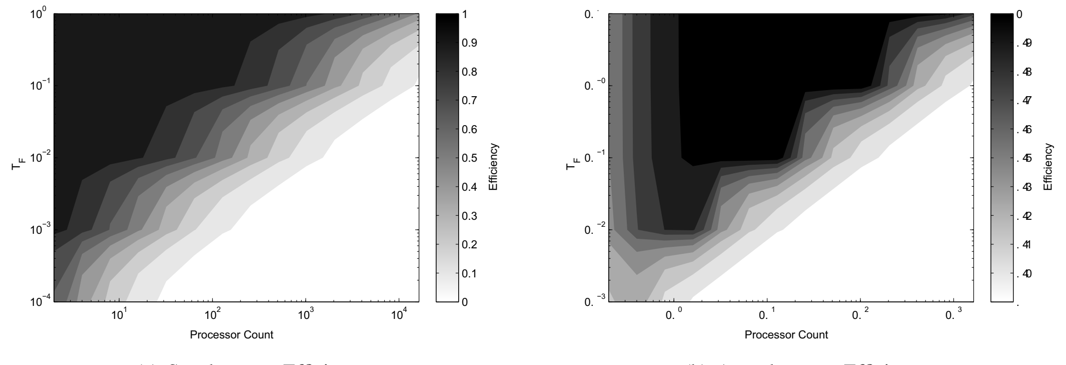


Figure 4. Parallel speedup to reach various hypervolume thresholds on the 5-objective UF11 problem. Higher hypervolume thresholds correspond to higher-quality results.



(a) Synchronous Efficiency (b) Asynchronous Efficiency

Figure 5. Predicted efficiency of a synchronous MOEA (using the model developed by Erick Cantú-Paz [5]) compared against the predicted efficiency of an asynchronous MOEA using the simulation model. T_F ranges from 0.0001 up to 1 second, and P ranges from 2 to $16,384$ processors. The coloring shows the efficiency, with highest efficiency in the red regions and worst efficiency in the blue regions. Note the log scale of the x- and y- axes.

evaluation. So, when T_F is highly-variable, we expect the efficiency of the synchronous model to decline while the asynchronous model remains unchanged.

VII. Conclusion

In this study, we analyzed the scalability of the Borg MOEA. We developed an analytical model of the parallel processing time and derived the processor count lower and upper bounds. This analytical model is limited by its inability to model the interactions between the master and worker nodes that introduce resource contention and additional overhead. To more accurately model the Borg MOEA, we developed a simulation model using the SimPy simulation package for Python. From this model, we can accurately model the parallel processing time, efficiency, and ideal processor count to maximize efficiency.

In addition to analyzing the speedup achieved by the Borg MOEA, we also analyzed the algorithm's dynamics at various processor counts. Our results indicate that the effectiveness of the asynchronous Borg MOEA's auto-adaptive search is strongly shaped by parallel scalability and problem difficulty. The algorithm's performance is maximized only when high parallel efficiency enable it to fully activate its auto-adaptive evolutionary operators when addressing problems of increasing difficulty. Our results indicate that the Borg MOEA and other asynchronous EAs are scalable, but the scalability is limited by several factors. To increase efficiency and improve algorithm dynamics on larger-scale parallel systems ($> 16,000$ processors), it will be necessary to transition to a more adaptive, island-based topology for the Borg MOEA. The development and analysis of an adaptive, island-based topology will be the subject of future work.

Acknowledgment

This work used the Extreme Science and Engineering Discovery Environment (XSEDE), which is supported by National Science Foundation grant number OCI-1053575.

The authors acknowledge the Texas Advanced Computing Center (TACC) at The University of Texas at Austin for providing HPC resources that have contributed to the research results reported within this paper.

References

[1] D. E. Goldberg, "Genetic and evolutionary algorithms come of age," *Communications of the ACM*, vol. 37, no. 3, pp. 113–119, 1994.

[2] T. Bäck, D. B. Fogel, and Z. Michalewicz, *Handbook of Evolutionary Computation*. Taylor & Francis, 1997.

[3] Y. Tang, P. M. Reed, and J. B. Kollat, "Parallelization strategies for rapid and robust evolutionary multiobjective optimization in water resources applications," *Advances in Water Resources*, vol. 30, pp. 335–353, 2007.

[4] M. P. Ferringer, D. B. Spencer, and P. Reed, "Many-objective reconfiguration of operational satellite constellations with the large-cluster epsilon Non-dominated Sorting Genetic Algorithm-II," in *IEEE Congress on Evolutionary Computation (CEC 2009)*, May 2009, pp. 340–349.

[5] E. Cantú-Paz, *Efficient and Accurate Parallel Genetic Algorithms*. Kluwer Academic Publishers, 2000.

[6] A. D. Bethke, "Comparison of genetic algorithms and gradient-based optimizers on parallel processors: Efficiency of use of processing capacity," University of Michigan, Ann Arbor, MI, Tech. Rep. Tech. Rep. No. 197, Logic of Computers Group, 1976.

[7] E. Cantú-Paz, "A survey of parallel genetic algorithms," *CALCULATEURS PARALLELES*, vol. 10, 1998.

[8] ——, "Designing efficienct master-slave parallel genetic algorithms," University of Illinois, Urbana, IL, Tech. Rep. IlliGAL Report No. 97004, 1997.

[9] E. Cantú-Paz and D. E. Goldberg, "Modeling idealized bounding cases of parallel genetic algorithms," in *Proceedings of the Second Annual Conference of Genetic Programming*. San Francisco, CA: Morgan Kaufmann, 1997.

[10] ——, "Predicting speedups of idealized bounding cases of parallel genetic algorithms," in *Proceedings of the Seventh International Conference on Genetic Algorithms*. San Francisco, CA: Morgan Kaufmann, 1997, pp. 113–121.

[11] C. A. Coello Coello, G. B. Lamont, and D. A. Van Veldhuizen, *Evolutionary Algorithms for Solving Multi-Objective Problems*. New York, NY, USA: Springer Science+Business Media, LLC, 2007.

[12] D. Hadka and P. Reed, "Borg: An auto-adaptive many-objective evolutionary computing framework," *Evolutionary Computation*, 2012.

[13] ——, "Diagnostic assessment of search controls and failure modes in many-objective evolutionary optimization," *Evolutionary Computation*, vol. 20, no. 3, pp. 423–452, 2012.

[14] D. Hadka, P. Reed, and T. Simpson, "Diagnostic assessment of the borg moea for many-objective product family design problems," in *WCCI 2012 World Congress on Computational Intelligence, Congress on Evolutionary Computation*, Brisbane, AUS, June 2012.

[15] P. M. Reed, D. M. Hadka, J. D. Herman, J. R. Kasprzyk, and J. B. Kollat, "Evolutionary multiobjective optimization in water resources: The past, present, and future," *Advances in Water Resources*, 2012.

[16] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 67–82, 1997.

[17] K. Deb and R. B. Agrawal, "Simulated binary crossover for continuous search space," Indian Institute of Technology, Kanpur, UP, India, Tech. Rep. IITK/ME/SMD-94027, Nov. 1994.

[18] R. Storn and K. Price, "Differential evolution — a simple and efficient heuristic for global optimization over continuous spaces," *Journal of Global Optimization*, vol. 11, no. 4, pp. 341–359, 1997.

[19] K. Deb, D. Joshi, and A. Anand, "Real-coded evolutionary algorithms with parent-centric recombination," *Computational Intelligence, Proceedings of the World on Congress on*, vol. 1, pp. 61–66, 2002.

[20] S. Tsutsui, M. Yamamura, and T. Higuchi, "Multi-parent recombination with simplex crossover in real coded genetic algorithms," in *Genetic and Evolutionary Computation Conference (GECCO 1999)*, 1999.

[21] H. Kita, I. Ono, and S. Kobayashi, "Multi-parental extension of the unimodal normal distribution crossover for real-coded genetic algorithms," in *Congress on Evolutionary Computation*, 1999, pp. 1581–1588.

[22] M. Laumanns, L. Thiele, K. Deb, and E. Zitzler, "Combining convergence and diversity in evolutionary multi-objective optimization," *Evolutionary Computation*, vol. 10, no. 3, 2002.

[23] J. Horn, "The nature of niching: Genetic algorithms and the evolution of optimal, cooperative populations," Ph.D. dissertation, University of Illinois, 1995.

[24] S. W. Mahfoud, "Niching methods for genetic algorithms," Ph.D. dissertation, University of Illinois, 1995.

[25] Y. Tang, P. Reed, and T. Wagener, "How effective and efficient are multiobjective evolutionary algorithms at hydrologic model calibration?" *Hydrology and Earth System Science*, vol. 10, pp. 289–307, 2006.

[26] Q. Zhang, W. Liu, and H. Li, "The performance of a new version of MOEA/D on CEC09 unconstrained MOP test instances," in *Congress on Evolutionary Computation (CEC 2009)*. Piscataway, NJ, USA: IEEE Press, 2009, pp. 203–208.

[27] K. Deb, L. Thiele, M. Laumanns, and E. Zitzler, "Scalable multi-objective optimization test problems," in *Congress on Evolutionary Computation (CEC 2002)*, 2002, pp. 825–830.

[28] Q. Zhang, A. Zhou, S. Zhao, P. N. Suganthan, W. Liu, and S. Tiwari, "Multiobjective optimization test instances for the CEC 2009 special session and competition," University of Essex, Tech. Rep. CES-487, 2009.

[29] Q. Zhang and P. N. Suganthan, "Final report on CEC'09 moea competition," in *Congress on Evolutionary Computation (CEC 2009)*. Piscataway, NJ, USA: IEEE Press, 2009.

[30] E. Zitzler, L. Thiele, M. Laumanns, C. M. Fonseca, and V. G. da Fonseca, "Performance assessment of multiobjective optimizers: An analysis and review," *IEEE Transactions on Evolutionary Computation*, vol. 7, no. 2, pp. 117–132, 2002.

[31] C. L. Bloebaum and A. M. R. McGowan, "Design of compex engineered systems," *Journal of Mechanical Design*, vol. 132, no. 12, pp. 1–2, 2010.