Effect of Global Parallelism on the Behavior of a Steady State Genetic Algorithm for Design Optimization

Khaled Rasheed
Department of Computer Science
Rutgers, The State University of New Jersey
New Brunswick, NJ 08903
krasheed@cs.rutgers.edu

Brian D. Davison
Department of Computer Science
Rutgers, The State University of New Jersey
New Brunswick, NJ 08903
davison@cs.rutgers.edu

Abstract-

In this paper we investigate the effect of global parallelism using a master slave approach, on the behavior of a steady state genetic algorithm for design optimization. Empirical results in several engineering design domains demonstrate that this simple form of parallelism which has the potential for almost linear speedup, does not significantly disturb the convergence pattern of the GA even when the number of processors is comparable to the size of the population.

1 Introduction

The abundance of powerful workstations at present makes parallelization an obvious enhancement to many optimization techniques, including genetic algorithms [Goldberg 1989, Michalewicz 1996]. Parallelism has the potential to improve GA performance through the use of alternative models of computation.

There are numerous methods for parallelizing a genetic algorithm. The main categories are:

- **Global parallelism:** In this method a master processor has access to the whole GA population (hence the name "global") and conducts selection and reproduction operations. The master then enlists the help of one or more slave processors to evaluate the fitness of the new individuals. Examples of global parallelism can be found in [Abramson and Abela 1992, Hauser and Männer 1994].

- **Coarse-grained parallelism:** In this method the population is divided into a number of subpopulations (also called *islands* or *demes*) equal to the number of processors. Each processor evolves its island separately and independently from the other islands, except for occasional migrations of good individuals from one island to another. The success of this type of parallel GA depends on making correct decisions regarding some critical factors such as migration rates and the topology (i.e. who migrates to where). Examples of coarse-grained parallelism can be found in [Levine 1994, Braun 1991].

- **Fine-grained parallelism:** In this method a large number of processors handle one or a very small number of individuals each. This model is suitable for massively parallel computers. Examples of fine-grained parallelism can be found in [Nishikawa and Tamaki 1991, Shapiro and Navetta 1994].

- **Hybrid methods:** These are hybrids of any of the above three methods. Examples include [Gorges-Schleuter 1991, Lin *et al.* 1997].

Parallelism can certainly reduce the total elapsed clock-time for a solution, but as a change in model of computation (either real or simulated), it can change the behavior and the convergence pattern of the GA. We applied global parallelism to GADO (Genetic Algorithm for Design Optimization) because among the parallelism methods, it is the easiest to implement. The effects of global parallelism on GADO were investigated in this paper.

In the following section we describe the architecture of GADO, as well as the global parallelism approach and its effect on the behavior model of GADO. In section 3 we describe the design optimization domains in which we conducted our experiments. Section 4 discusses the results of our experiments conducted in several design optimization domains in order to examine the effect of global parallelism on the behavior (especially the convergence pattern) of GADO. The paper is concluded in section 5.

2 GA Architecture

2.1 GADO: Genetic Algorithm for Design Optimization

GADO, the GA used in this research, is described in detail in [Rasheed 1998]. Each individual in the GA population represents a parametric description of an artifact, such as an aircraft, or a process, with each parameter taking on a value in some continuous interval. The fitness of each individual is based on the sum of a proper measure of merit computed by a simulator (such as the takeoff mass of an aircraft), and a penalty function if relevant (such as to impose legal limits on the permissible noise of an aircraft). Operators are applied to elements of the population via some selection scheme. Here selection was performed by rank (rather than via the actual value of the fitness function on each individual) because of the wide range of fitness values caused by the use of a penalty function — rank selection prevents the first discovered evaluable/feasible points from dominating the population. A steady

state GA model is used, in which existing points in the population are replaced by newly generated points via some replacement strategy. The replacement strategy used here takes into consideration both the fitness and the proximity of the points in the GA population, with the goal of selecting for replacement a point that both has low fitness and is relatively close to the point being introduced. The GA stops when either the maximum number of evaluations has been exhausted or the population completely loses diversity and practically converges to a single point in the search space. The GA architecture also includes a screening module (SM) [Rasheed and Hirsh 1997, Rasheed and Hirsh 1999] and a diversity maintenance module (DMM) which can be turned on or off. The SM saves time by preventing the GA from evaluating points that are close to previously encountered bad points, through the use of a case based learning technique. The DMM maintains diversity by preventing the GA from evaluating candidate points which are extremely close to previously evaluated points. The DMM also attempts to restore diversity through a re-seeding operation if severe loss of diversity is detected in the early stages of the optimization.

2.2 Global parallelism

In global parallelism there is a master processor that manipulates the GA population and performs the selection and reproduction operations, and a number of slave processors that evaluate the fitness of new individuals.

In the case of generational GAs, the global parallel GA is called *synchronous* if the master waits for the slaves to finish evaluating an entire generation before generating any individuals of the following generation. A synchronous global parallel GA thus follows exactly the same trace as its serial counterpart except that it probably executes faster. If the master does not necessarily wait, the GA is called *asynchronous*, does not retain the serial behavior and resembles a generational GA with a generation gap. The speedup of synchronous global parallel GAs has been analyzed in [Cantu-Paz 1998].

In this paper, however, we are interested in applying global parallelism to a steady state GA (GADO). In this case the master generates new individuals one at a time (through selection and crossover and mutation) and then sends these individuals to slave processors to be evaluated. After a short initialization period, the master will continuously monitor its slaves. Every time a slave finishes evaluating an individual, the master will insert this individual into the GA population (possibly replacing another individual) and then the master will generate a new individual and send it to that same slave to evaluate it. In our implementation, we have the option to make the master simultaneously treat itself as another slave. This is a useful approach when the number of processors is limited and/or the evaluation time of an individual is very large compared to the communication time between the master and slaves (which is common in the area of engineering design optimization in which the fitness evaluation typically takes several seconds to several hours).

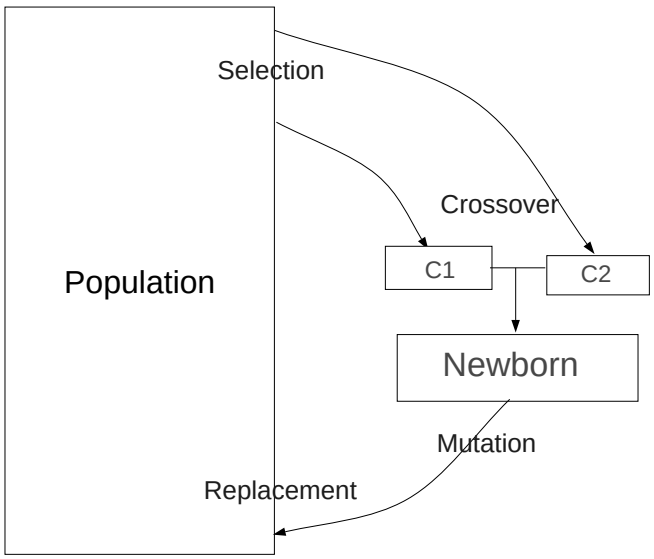

Figure 1: In the steady state model for genetic algorithms, a single new individual is created and then inserted into the population before repeating the process.

Assuming that all processors (including the master) have equal speeds, it can be shown that the speedup achievable by this approach would be:

$$Speedup = \frac{(S+1) \cdot T_e}{T_e + S \cdot T_c}$$

Where
S = number of slaves (not including the master)
T_e = fitness evaluation time of an individual
T_c = communication time between the master and a slave

If the master does not treat itself as a slave the speedup would be:

$$Speedup = \frac{S \cdot T_e}{T_e + T_c}$$

The above equations hold for as long as $T_e \geq S \cdot T_c$ and the analysis becomes more complicated otherwise. This analysis ignores the effect of parallelism on the behavior of the GA. This effect will be studied in the remainder of this paper.

2.3 Application of global parallelism to GADO

In this system, there is a master processor/process running GADO, and a number of slave processors/processes that simulate or evaluate potential designs. These simulations can be spread across many machines to get close to linear speed-up when the time for evaluating a design is sufficiently large.

In GADO, even this simple implementation of parallelism may have adverse effects. GADO was designed (and tested) for steady-state operation, in which for each iteration an ex-

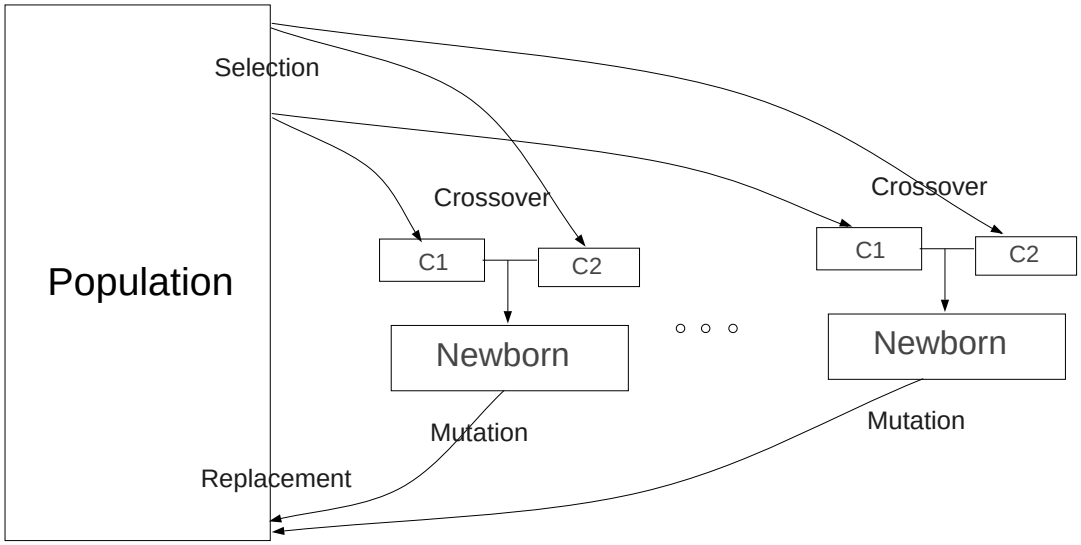

Figure 2: When parallelized, GADO must create multiple individuals and evaluate them in parallel.

isting member of the population is replaced by a new individual based on some replacement strategy (see Figure 1). Many other GAs use a generational approach, in which the entire population is replaced by a new population, and so are more amenable to parallelization.

In order to fit into a parallel architecture, GADO had to be modified to allow more than one individual to be created and simulated at a time. This means that the creation of a new individual may not be affected by individuals created one or two steps ago because they have not yet been placed into the population (see Figure 2). Even worse, GADO may get back individuals in a different order than originally created, as some processors/processes may complete their evaluations faster than others (such as a result of heterogeneous processing environments, external loads, etc.).

To investigate this issue, GADO was modified so that the number of slaves could be varied and allow or not allow random return ordering. Optimization experiments on one modern realistic engineering design domain as well as eight engineering design benchmark domains were run. For each experiment, optimizations for a number of slaves equal to 0, 1, 5, 10, 50 and 100 were attempted.

3 Experimental Domains

3.1 Supersonic Transport Aircraft Design

Our first domain concerns the conceptual design of supersonic transport aircraft. We summarize it briefly here; it is described in more detail in [Gelsey *et al.* 1996]. Figure 3 shows a diagram of a typical airplane automatically designed by our software system. The GA attempts to find a good design for a particular mission by varying the aircraft conceptual design parameters in Table 1 over a continuous range of values.

Table 1: Aircraft Parameters to Optimize

No.	*Parameter*
1	exhaust nozzle convergent length
2	exhaust nozzle divergent length
3	exhaust nozzle external length
4	exhaust nozzle radius(r7)
5	engine size
6	wing area
7	wing aspect ratio
8	fuselage taper length
9	effective structural t/c
10	wing sweep over design mach angle
11	wing taper ratio
12	Fuel Annulus Width

The GA evaluates candidate designs using a multidisciplinary simulator. In our current implementation, the GA's goal is to minimize the takeoff mass of the aircraft, a measure of merit commonly used in the aircraft industry at the conceptual design stage. Takeoff mass is the sum of fuel mass, which provides a rough approximation of the operating cost of the aircraft, and "dry" mass, which provides a rough approximation of the cost of building the aircraft. A complete mission simulation requires about 1/5 second of CPU time on a DEC Alpha 250 4/266 desktop workstation.

The aircraft simulation model used is based on both implicit and explicit assumptions and engineering approximations and since it is being used by a numerical optimizer

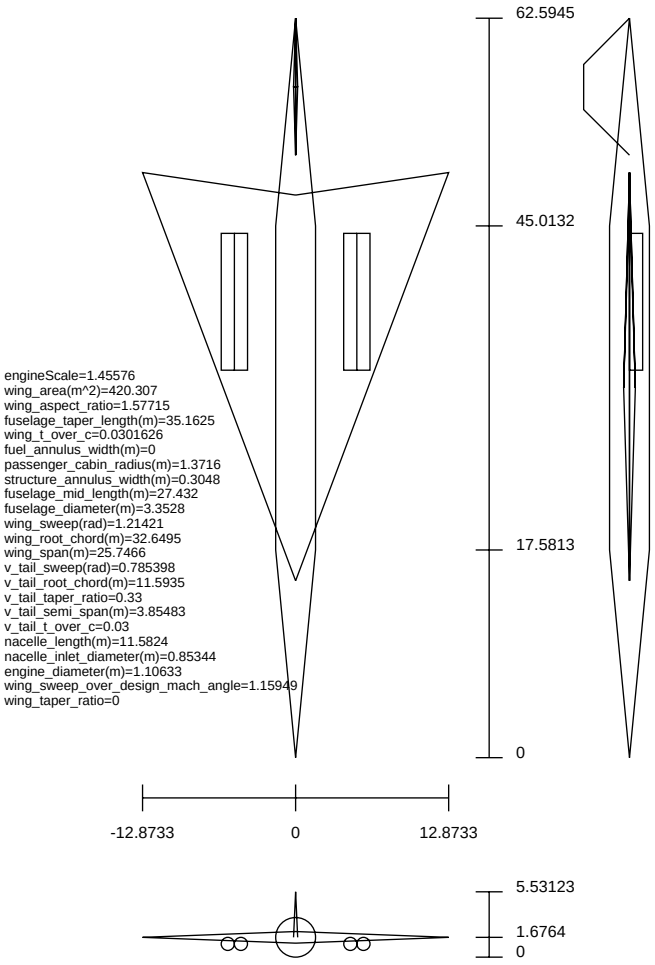

Figure 3: Supersonic transport aircraft designed by our system (dimensions in feet)

rather than a human domain expert, some design parameter sets may correspond to aircrafts that violate these assumptions and therefore may not be physically realizable even though the simulator does not detect this fact. For this reason a set of constraints has been introduced to safeguard the optimization process against such violations. In particular, a penalty function approach was used to incorporate the effect of constraint violations into the optimization: the penalty function was added to the takeoff mass value returned by the simulator and the resulting sum was the quantity that the optimizer actually minimizes (which therefore also serves as the fitness value assigned to each point of the GA population). The specific penalty function was simply a large constant multiplied by the sum of the amounts of constraint violation for all the violated constraints.

The presence of constraints induced a partition of the search space into three mutually exclusive regions:

1. Unevaluable points: These are points that represent designs that violate the model assumptions so much that the simulator cannot complete the simulation process to produce any significant information. For such points a very large fictitious takeoff mass is generated as the value of the objective function.

2. Infeasible evaluable points: These are points that represent unrealizable aircrafts but the type and extent of model violation is moderate enough for the simulator to complete its work and report the constraint violation information. As described above, a penalty function is added to the takeoff mass returned by the simulator to account for the model violation.

3. Feasible points: The simulator succeeds in evaluating the take off mass for such points and no violations occur in the process. The penalty function for these points is zero.

Table 2: Benchmark domains

Domain *No.*	Sandgren *No.*	*Dim.*	**Constraints** *inequ.*	*equ.*	best f
1	13	5	4	0	26.78
2	2	3	2	0	-3.3
3	3	5	6	0	-3.06
4	8	3	2	0	-5.68
5	6	6	0	4	8.92
6	15	16	0	8	244.8
7	21	13	13	0	97.5
8	22	16	19	0	174.7

3.2 Benchmark domains

These problems were first introduced by Eric Sandgren in his Ph.D. thesis [Sandgren 1977] and have since been used in engineering design optimization research as benchmarks. One of the recent experiments involving these domains was reported in [Powell and Skolnick 1993], in which a GA package called OOGA and a numerical optimization package called NumOpt were compared to each other in 10 of Sandgren's domains. The 10 domains were a representative sample of the original 30. We ran experiments in eight of these 10 domains.[1] All eight were minimization problems.

Some properties of the benchmarks used are summarized in Table 2. The second column of the table shows the problem numbers as they appeared in Sandgren's thesis. The third column shows the problem dimensions (i.e., the number of design variables in each problem). The fourth and fifth columns show the number of inequality and equality constraints respectively. The sixth column shows the best known optima of the problems. A detailed description of these domains is given in [Sandgren 1977]. A detailed description of the experiments comparing GADO to other optimizers in these domains can be found in [Rasheed 1998]. In this paper we only

[1] We were unable to do any comparison in 2 of the 10 domains because they had unbounded variables.

focus on studying the effect of global parallelism on GADO's behavior.

4 Results and Discussion

4.1 Supersonic Transport Aircraft Design

Ten random populations of 120 points each were generated, and for each population the GA was allowed to proceed for 12000 iterations (an iteration denotes one call to the simulator, which takes, on average, 0.2 seconds) once with number of slaves equal to 0[2], 1, 5, 10, 50 and 100. The results of our experiment are shown in Figure 4. The graph plots the measure of merit (takeoff mass) of the best point found so far against the number of iterations. Each curve represents the average of the best individuals seen so far of the runs from the ten starting populations. The solid curve represents the average of the ten runs of GADO without parallelism and the dotted curves represent the averages of the sets of runs with parallelism. The graph shows that the effect of parallelism on performance was minimal in all cases.

The above experiments make the assumption of a first-in, first-out evaluation by the slaves. To simulate the effects of real-world non-determinism on the order of parallel evaluation returns, a result was collected from a randomly selected position in the list of slaves with uniform probability and then inserted back into the GA population. The experiments were then re-run and the results are shown in Figure 5. The results are the same as the non-randomized case.

4.2 Benchmark domains

Five random populations for each domain were generated and averaged for a number of slaves equal to 0 (serial case), 1, 5, 10, 50 and 100. The results can be found in Figure 6 through Figure 19. We found that the performance did not change significantly in all domains[3] and under all setups (with or without parallelism, different numbers of slaves, random or ordered return).

In general, randomized runs with 100 slaves often performed slightly worse, but the other runs had averages very similar to the averages of the serial versions.

Depending on the domain, the experiments used population sizes of 20-160 individuals, and ran for 1000-50000 iterations. Thus, the numbers of slaves considered are potentially significant fractions or sometimes multiples of the population size. Even so, only at 100 slaves did degradation in performance in some domains start becoming noticeable.

[2]We used the approach of making the master treat itself also as one of the slaves. Thus 0 slaves means the master does all the work (serial case). one slave means two processors actually do evaluations, and so on.

[3]We had to exclude benchmark 6 (Sandgren's problem 15) from our final analysis as several runs under all different setups did not find any feasible points. We were unable to do any qualitative comparisons.

5 Conclusion

In this paper the effect of global parallelism on the behavior of GADO, a steady state GA used for design optimization was experimentally studied. The global parallelism approach is a simple form of parallelism in which a master processor maintains the GA population and executes the selection and reproduction operations. The master then relies on slave processors to evaluate the fitnesses of the new individuals. Global parallelism has the potential for close to linear speedup in domains with very expensive fitness evaluations — such as those which arise in the field of engineering design optimization. Moreover it is easy to implement or add to an existing GA.

Our apprehension before we did this study was that global parallelism would have a strong impact on the behavior of GADO, being a steady state GA. We were concerned that as the number of slave processors becomes significantly large the deviation from the steady state model — in which each new individual is generated after the previous one has already been evaluated and possibly inserted into the GA population — may degrade the performance and make it more like that of a random search. In contrast to our expectations, we found that the degradation in performance was quite limited and was only noticeable when the number of slaves was in the order of 100 which is larger than most population sizes used in this study. Even when we changed the order of return of the individuals being evaluated by the slaves — as a means of simulating a real world scenario with different work loads and/or speeds for different processors — the results were almost unaffected.

We conclude that global parallelism is a competitive method of parallelism even when the number of processors is comparable to the population size. We speculate that for much larger numbers of processors, hybrid approaches for parallelism are likely to be more efficient.

Acknowledgments

We thank Donald Smith, and Keith Miyake for their invaluable assistance in this research. We also thank all members of the HPCD project, especially Haym Hirsh. This research was part of the Rutgers-based HPCD (Hypercomputing and Design) project supported by the Advanced Research Projects Agency of the Department of Defense through contract ARPA-DABT 63-93-C-0064, and by the Rutgers Center for Computational Design (CCD).

Bibliography

[Abramson and Abela 1992] D. Abramson and J. Abela. A parallel GA for solving the school timetabling problem. In *Proceedings of the Fifteenth Australian Computer Science Conference (ACSC-15)*, 1992.

[Braun 1991] H. Braun. On solving travelling salesman problems by genetic algorithms. In H.-P. Schwefel and

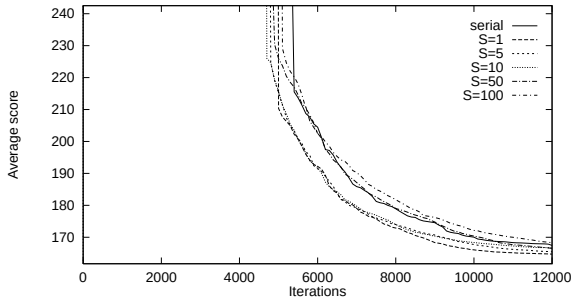

Figure 4: Optimization runs on the aircraft design domain for various numbers of slaves without randomized return.

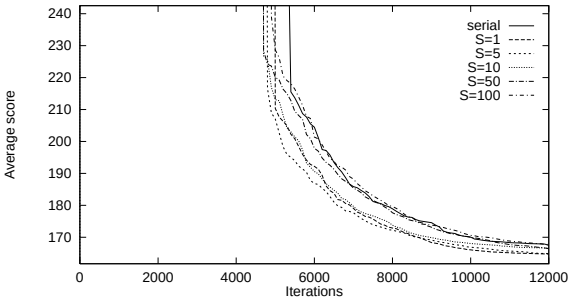

Figure 5: Optimization runs on the aircraft design domain for various numbers of slaves with randomized return.

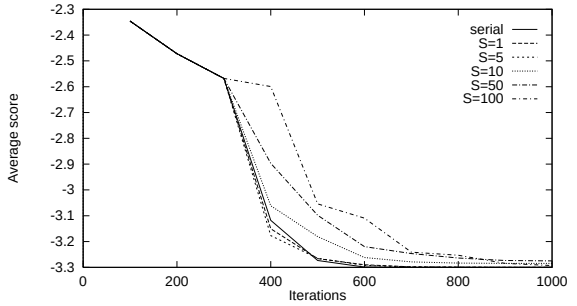

Figure 6: Optimization runs on Sandgren benchmark 2 for various numbers of slaves without randomized return.

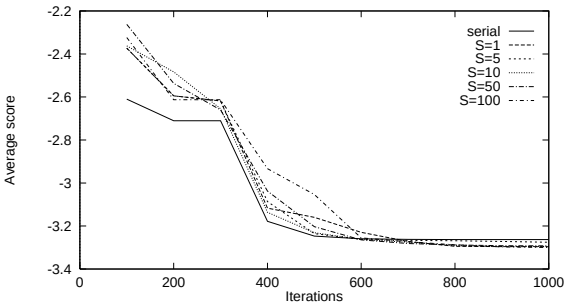

Figure 7: Optimization runs on Sandgren benchmark 2 for various numbers of slaves with randomized return.

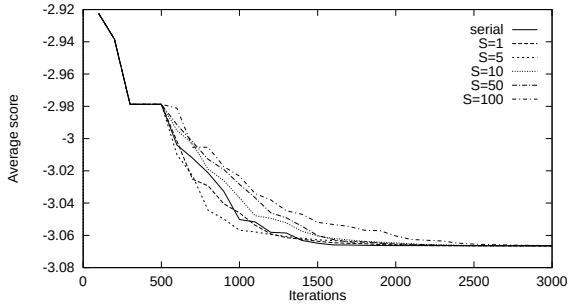

Figure 8: Optimization runs on Sandgren benchmark 3 for various numbers of slaves without randomized return.

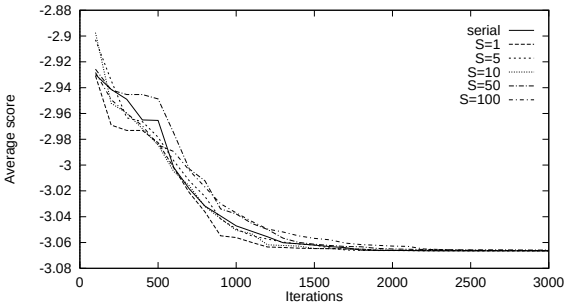

Figure 9: Optimization runs on Sandgren benchmark 3 for various numbers of slaves with randomized return.

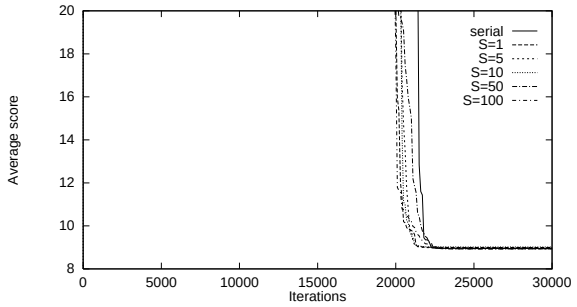

Figure 10: Optimization runs on Sandgren benchmark 6 for various numbers of slaves without randomized return.

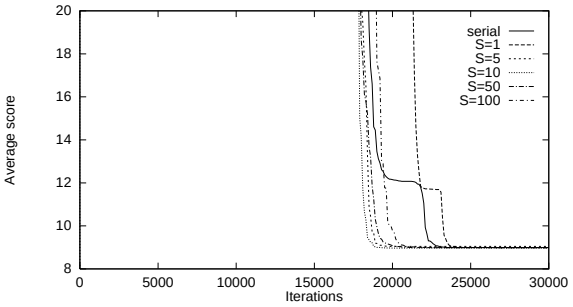

Figure 11: Optimization runs on Sandgren benchmark 6 for various numbers of slaves with randomized return.

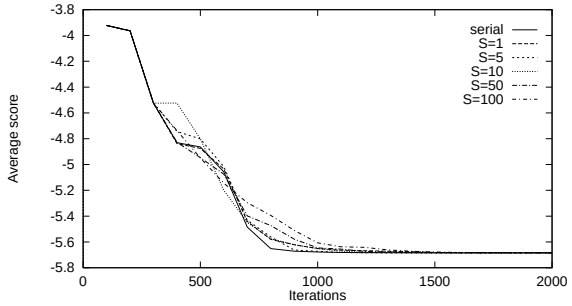

Figure 12: Optimization runs on Sandgren benchmark 8 for various numbers of slaves without randomized return.

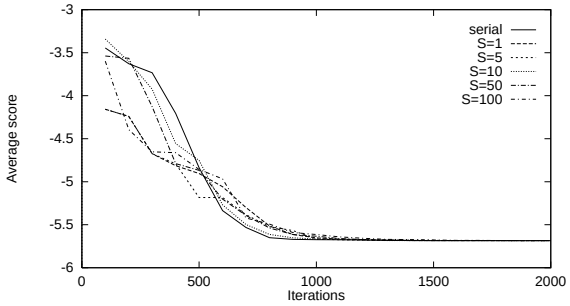

Figure 13: Optimization runs on Sandgren benchmark 8 for various numbers of slaves with randomized return.

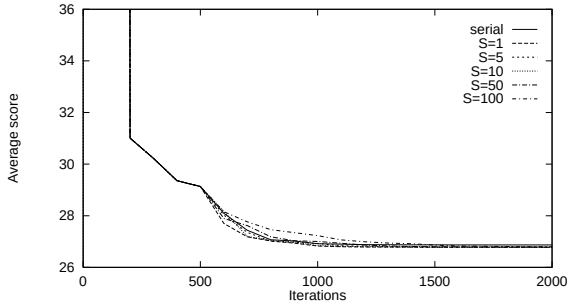

Figure 14: Optimization runs on Sandgren benchmark 13 for various numbers of slaves without randomized return.

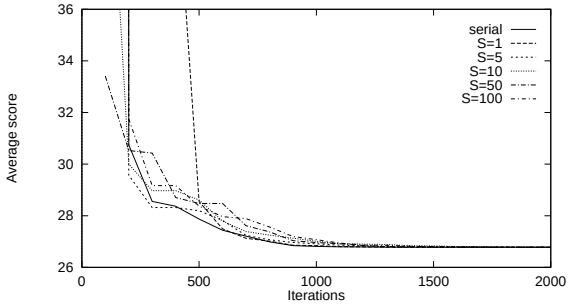

Figure 15: Optimization runs on Sandgren benchmark 13 for various numbers of slaves with randomized return.

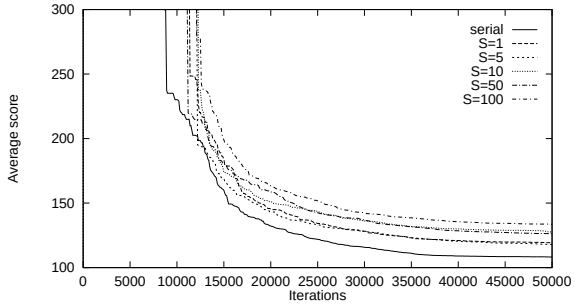

Figure 16: Optimization runs on Sandgren benchmark 21 for various numbers of slaves without randomized return.

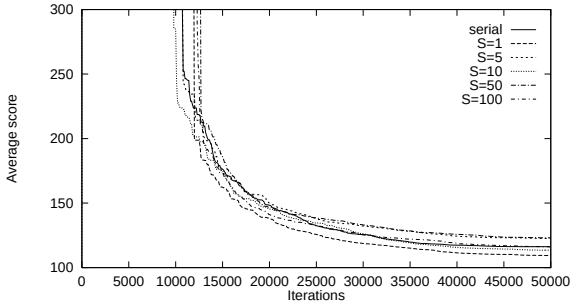

Figure 17: Optimization runs on Sandgren benchmark 21 for various numbers of slaves with randomized return.

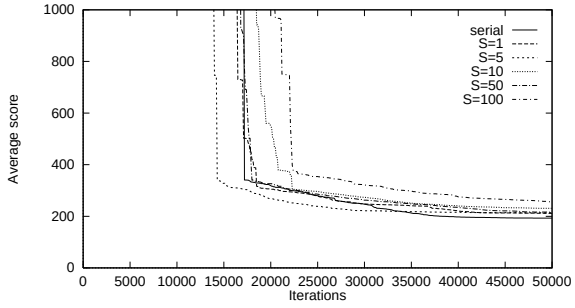

Figure 18: Optimization runs on Sandgren benchmark 22 for various numbers of slaves without randomized return.

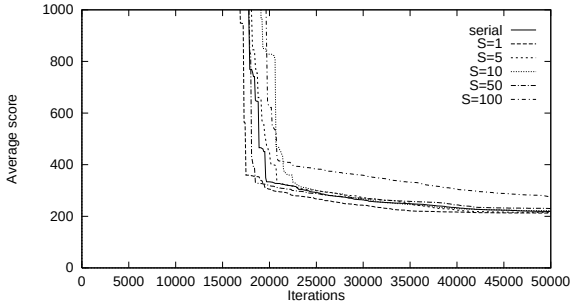

Figure 19: Optimization runs on Sandgren benchmark 22 for various numbers of slaves with randomized return.

R. Männer, editors, *Parallel Problem Solving from Nature - Proceedings of 1st Workshop, PPSN 1*, volume 496 of *Lecture Notes in Computer Science*, pages 129–133, Dortmund, Germany, 1-3 October 1991. Springer-Verlag, Berlin, Germany.

[Cantu-Paz 1998] Erick Cantu-Paz. Designing efficient master-slave parallel genetic algorithms. In *Genetic Programming 1998: Proceedings of the Third Annual Conference*, page 455, University of Wisconsin, Madison, Wisconsin, USA, 22-25 July 1998. Morgan Kaufmann.

[Gelsey *et al.* 1996] Andrew Gelsey, M. Schwabacher, and Don Smith. Using modeling knowledge to guide design space search. In *Fourth International Conference on Artificial Intelligence in Design '96*, 1996.

[Goldberg 1989] David Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Reading, Mass., 1989.

[Gorges-Schleuter 1991] Gorges-Schleuter. ASPARAGOS: A parallel genetic algorithm for population genetics. In J. D. Becker, I. Eisele, and F. W. Mundemann, editors, *Parallelism, Learning, Evolution. Workshop on Evolutionary Models and Strategies - WOPPLOT 89*, pages 407–418, Berlin, 1991. Springer-Verlag.

[Hauser and Männer 1994] R. Hauser and R. Männer. Implementation of standard genetic algorithms on MIMD machines. In Y. Davidor and H.-P. Schwefel, editors, *Parallel Problem Solving From Nature – PPSN III*, volume 866 of *Lecture Notes in Computer Science*, pages 504–513, Berlin, 1994. Springer Verlag.

[Levine 1994] David Levine. A parallel genetic algorithm for the set partitioning problem. Technical Report ANL-94/23, Argonne National Laboratory, Mathematics and Computer Science Division, May 1994.

[Lin *et al.* 1997] Shyh-Chang Lin, Erik D. Goodman, and William F. Punch. A genetic algorithm approach to dynamic job shop scheduling problems. In Thomas Bäck, editor, *Proceedings of the 7th International Conference on Genetic Algorithms*, pages 481–488, San Francisco, July19–23 1997. Morgan Kaufmann.

[Michalewicz 1996] Zbigniew Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag, New York, 1996.

[Nishikawa and Tamaki 1991] Y. Nishikawa and H. Tamaki. A genetic algorithm as applied to the jobshop scheduling. *Transactions of the Society of Instrument and Control Engineers*, 27(5):593–599, May 1991. In Japanese.

[Powell and Skolnick 1993] D. Powell and M. Skolnick. Using genetic algorithms in engineering design optimization with non-linear constraints. In *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 424–431. Morgan Kaufmann, July 1993.

[Rasheed and Hirsh 1997] Khaled Rasheed and Haym Hirsh. Using case-based learning to improve genetic-algorithm-based design optimization. In *Proceedings of the Seventh International Conference on Genetic Algorithms*. Morgan Kaufmann, 1997.

[Rasheed and Hirsh 1999] Khaled Rasheed and Haym Hirsh. Learning to be selective in genetic-algorithm-based design optimization. *Artificial Intelligence in Engineering, Design, Analysis and Manufacturing*, 1999. To appear.

[Rasheed 1998] Khaled Rasheed. GADO: A genetic algorithm for continuous design optimization. Technical Report DCS-TR-352, Department of Computer Science, Rutgers University, New Brunswick, NJ, January 1998. Ph.D. Thesis, http://www.cs.rutgers.edu/∼krasheed/thesis.ps.

[Sandgren 1977] Eric Sandgren. The utility of nonlinear programming algorithms. Technical report, Purdue University, 1977. Ph.D. Thesis.

[Shapiro and Navetta 1994] B. A. Shapiro and J. Navetta. A massively parallel genetic algorithm for RNA secondary structure prediction. *The Journal of Supercomputing*, 8(3):195–207, November 1994.