Effect of Global Parallelism on a Steady State GA[1]

Brian D. Davison **Khaled Rasheed**
Department of Computer Science
Rutgers, The State University of New Jersey
New Brunswick, NJ 08903
{davison,krasheed}@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.

1 Introduction

The abundance of powerful workstations at present makes parallelization an obvious enhancement to many optimization techniques, including genetic algorithms. 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.
- **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).
- **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.
- **Hybrid methods:** These are combinations of any of the above three methods.

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.

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 individuals via a rank-based selection scheme. 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.

[1] The full version of this paper is available as a conference paper [Rasheed and Davison, 1999].

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).

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 [Rasheed and Davison, 1999].

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 existing member of the population is replaced by a new individual based on some replacement strategy. 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. 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 various numbers of slaves were attempted.

3 Experimental Domains

3.1 Supersonic Transport Aircraft Design

Our first domain concerns the conceptual design of supersonic transport aircraft. It is summarized briefly in the main paper. The GA attempts to find a good design for a particular mission by varying the aircraft conceptual design parameters over a continuous range of values. Candidate designs are evaluated 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.

3.2 Benchmark domains

These problems were first introduced and described by Eric Sandgren in his Ph.D. thesis [Sandgren, 1977] and have since been used in engineering design optimization research as benchmarks. We ran experiments in eight of these domains. All were minimization problems.

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 num-

ber of slaves equal to zero[2], 1, 5, 10, 50 and 100. The results of our experiment are shown in Figure 1. 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 2. 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. Samples of our results can be found in Figure 3 through Figure 8. We found that the performance did not change significantly in all domains. 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.

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 initial apprehension 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 even when the number of processors is comparable to the population size.

Acknowledgments

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).

References

[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, Madison, WI, 22-25 July 1998. Morgan Kaufmann.

[Rasheed and Davison, 1999] Khaled Rasheed and Brian D. Davison. Effect of global parallelism on the behavior of a steady state genetic algorithm for design optimization. In *Proceedings of the Congress on Evolutionary Computation (CEC99)*, Washington, DC, July 1999.

[Rasheed, 1998] Khaled Rasheed. GADO: A genetic algorithm for continuous design optimization. Technical Report DCS-TR-352, Department of Computer Science, Rutgers, The State University of New Jersey, 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.

[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.

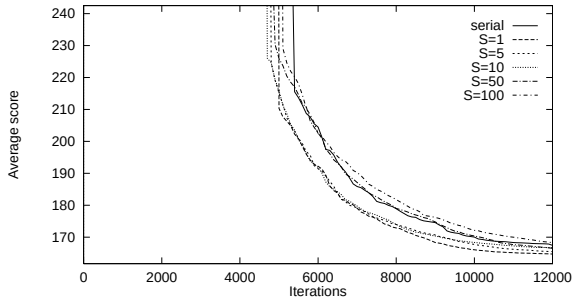

Figure 1: Optimization runs on the aircraft design domain for various numbers of slaves without randomized return.

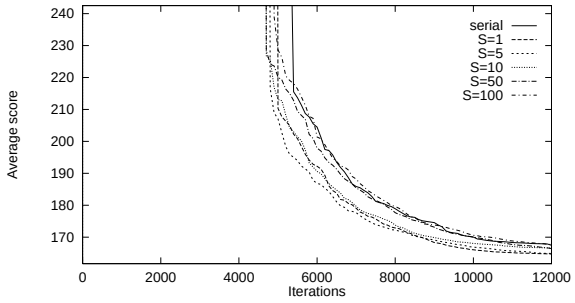

Figure 2: Optimization runs on the aircraft design domain for various numbers of slaves with randomized return.

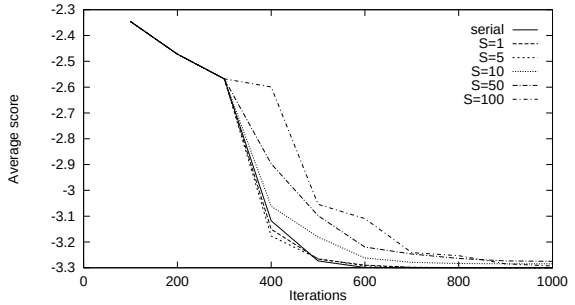

Figure 3: Optimization runs on Sandgren benchmark 2 for various numbers of slaves without randomized return.

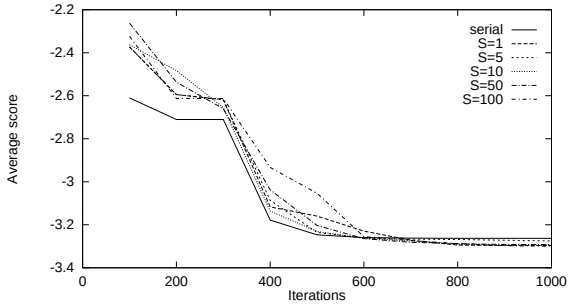

Figure 4: Optimization runs on Sandgren benchmark 2 for various numbers of slaves with randomized return.

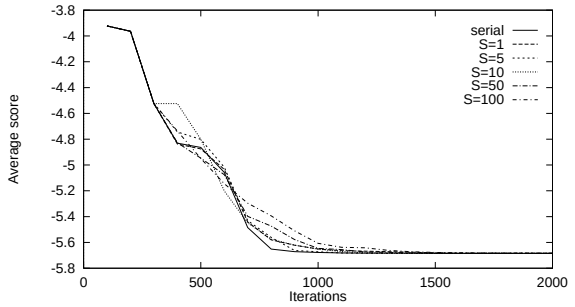

Figure 5: Optimization runs on Sandgren benchmark 8 for various numbers of slaves without randomized return.

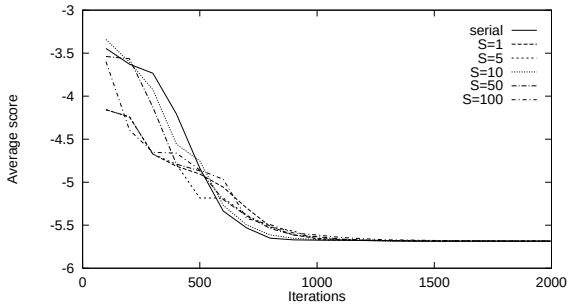

Figure 6: Optimization runs on Sandgren benchmark 8 for various numbers of slaves with randomized return.

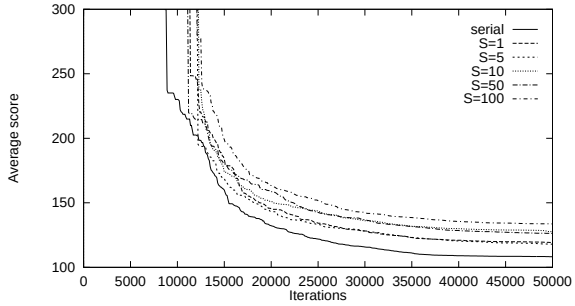

Figure 7: Optimization runs on Sandgren benchmark 21 for various numbers of slaves without randomized return.

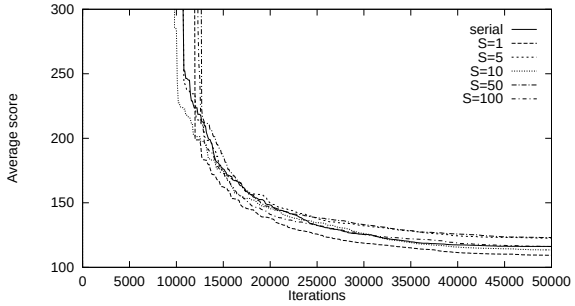

Figure 8: Optimization runs on Sandgren benchmark 21 for various numbers of slaves with randomized return.