RECEIVED

FEB 12 1996

OSTI

SAND96-0339C

CONF-960450--2

Analysis of the Numerical Effects of Parallelism on a Parallel Genetic Algorithm

William E. Hart* Scott Baden† Richard K. Belew‡ Scott Kohn‡

September 18, 1995

Abstract

This paper examines the effects of relaxed synchronization on both the numerical and parallel efficiency of parallel genetic algorithms. We describe a coarse-grain geographically structured parallel genetic algorithm. Our experiments show that asynchronous versions of these algorithms have a lower run time than synchronous GAs. Furthermore, we demonstrate that this improvement in performance is partly due to the fact that the numerical efficiency of the asynchronous genetic algorithm is better than the synchronous genetic algorithm. Our analysis includes a critique of the utility of traditional parallel performance measures for parallel GAs, and we evaluate the claims made by several researchers that parallel GAs can have superlinear speedup.

*Sandia National Laboratories, P. O. Box 5800, Albuquerque, NM 87185-1110; wehart@cs.sandia.gov

†Computer Science and Engineering Department, University of California - San Diego, La Jolla, CA 92093

DISCLAIMER

**Portions of this document may be illegible
in electronic image products. Images are
produced from the best available original
document.**

1 Introduction

Genetic algorithms (GAs) are stochastic search algorithms that have been successfully applied to a variety of optimization problems [5, 7]. Unlike most other optimization procedures, GAs maintain a *population of individuals* (set of solutions) that are competitively selected to generate new candidates for the global optima. Parallel GAs have been developed for a variety of parallel architectures [4, 9, 16]. These GAs have been used to solve a variety of difficult optimization problems, and the algorithmic modifications needed to efficiently parallelize GAs have provided insight into the dynamics of GAs' search [8].

Several arguments are commonly used to justify parallelism in GAs. First, parallelism can reduce the cost of function evaluations, which may dwarf other costs in the optimization process. Second, parallelism enables the use of larger population sizes. Empirical evidence suggests that large population sizes are needed to reliably solve certain types of difficult optimization problems. Here, parallelism is used to distribute the cost of the additional function evaluations needed for the large population size.

Another argument used to justify parallelism in GAs is that parallel GAs improve the quality of the solution found. Serial GAs have been observed to "prematurely converge" to nonoptimal solutions. Many parallel GAs are designed in such a way that competitive selection is localized between subsets of the total population. A consequence of this localization is that parallel GAs often reliably converge to optimal solutions.

The last two arguments for parallelism in GAs concern the observed superior numerical performance of these GAs. Gordon and Whitley [8] make a similar observation and argue that the algorithmic structure of parallel GAs is of interest, independent from their

implementation on a particular piece of hardware. They experimentally compared the performance of several parallel GAs, all of which were emulated on a sequential architecture. They observed that the emulations of parallel GAs were often more efficient than many serial GAs.

This paper complements the work of Gordon and Whitley by carefully examining the effects of relaxed synchronization on both the numerical efficiency and parallel efficiency of a parallel GA. Numerical efficiency measures the numerical costs associated with the GAs search, while the parallel efficiency measures the costs associated with the parallelization of the GA. We demonstrate that asynchronous parallel GAs can have a lower run time than synchronous parallel GAs. Further, we show that this improvement in performance is partly due to an improvement in the numerical efficiency of the asynchronous parallel GA. While many authors have noted that asynchronous parallel GAs typically have high hardware utilization of the processors, to our knowledge our analysis is the first to distinguish between hardware utilization and computation effort in parallel GAs.

We identify several factors of parallel GAs that make it difficult to determine their parallel performance. In this context, we evaluate a variety of performance measures and conclude that traditional measures of parallel efficiency are not particularly meaningful for these algorithms. Consequently, the run time is identified as the most useful measure for comparing the performance of two parallel GAs. Our analysis provides a basis for evaluating the performance of other parallel GAs. As an example, we evaluate the claims made by several researchers that parallel GAs can have superlinear speedup.

Section 2 provides background information on GAs. Section 3 describes the parallel GAs that we analyze, and outlines the experimental setup. Section 4 discusses properties

of GAs and their effect on performance measures for parallel GAs. Section 5 describes our experimental analysis, and we conclude with a discussion in Section 6.

2 Background

2.1 Genetic Algorithms

This paper considers GAs that minimize functions of the form $f : D \rightarrow \mathbf{R}$, where D is a compact subset of $\mathbf{R}^n$. The aim is to find x^* such that $f(x^*) = \min_{x \in D} f(x)$. Let x_i^t be the i th solution in the population at iteration t , so $x_i^t \in \mathbf{R}^n$. Figure 1 describes the general process a GA.

Competitive selection determines which solutions in the current population will be used to generate solutions for the next iteration. A standard selection mechanism is *proportional selection*, which stochastically selects individual i with probability $p_i^t = f(x_i^t) / \sum_j f(x_j^t)$, where x_i^t is the i th individual and $f(x_i^t)$ is the *fitness* of the individual - the value of the objective function at x_i . This selection method assumes that the GA is maximizing $f(x)$ and that $f(x) \geq 0$ for all x , but it can be easily modified to perform selection when minimizing or when the fitnesses are negative. A related selection mechanism is *rank selection* which stochastically selects x_i^t with a probability that is linearly related to the rank of $f(x_i^t)$ in the population.

New individuals are generated by applying *genetic operators* that take one or more individuals from the population and generate one or more new individuals. The two operators commonly used in GAs are mutation and crossover. Figure 2 illustrates the types of new

Initialize population (with randomly generated solutions)

Repeat $t = 1, 2 \dots$

Evaluate solutions in the population

Perform competitive selection

Apply genetic operators

Perform local search (optional)

Until convergence criteria satisfied

Figure 1: Pseudo-code for a genetic algorithm.

solutions generated by these two operators. Crossover combines pieces of two solutions to generate a new solution, and it is typically applied with high frequency. A variety of crossover operators have been developed for GAs that search in $\mathbf{R}^n$ [6]. Mutation makes a small change to a solution, and it is typically used with low frequency. For example, mutation might add a normally distributed random value to a single dimension of a solution.

2.2 Geographically Structured Genetic Algorithms

GAs can be distinguished by the manner in which competitive selection is performed. *Geographically structured GAs* (GSGAs) perform a structured selection in which solutions compete against a fixed subset of the population that partially overlaps with subsets used by other individuals. Thus the crossover operator is applied to two individuals selected from a subset of the population. *Panmictic* GAs perform a global competitive selection that allows any pair of individuals to be selected for the crossover operator. An example of panmictic

Figure 2: Examples of how mutation and crossover generate a solutions. Colored bars illustrate what subsections of the solutions are pieced together and modified: (a) solution generated by mutation, (b) solution generated by crossover, and (c) solution generated by both mutation and crossover.

GAs are those that use proportional selection.

Parallel GSGAs are typically implemented as *fine grain* algorithms in which each processor is responsible for evaluating and processing a single individual. Among fine grain GSGAs, two classes of algorithms can be distinguished: GSGAs designed for MIMD architectures [9] and GSGAs designed for SIMD architectures [4, 13, 16]. The method of structuring the competitive selection differs between these two classes based on differences in the hardware that are exploited by these algorithms to achieve high utilization. For example, the most common method of structuring the competitive selection for SIMD fine grain GSGAs uses a toroidal grid like the one in Figure 3 [4, 13, 16]. Every individual in the population is assigned to a location on the grid in an arbitrary manner.

The grid is used to determine subsets of solutions that compete for a location on the grid. There are distinct notions of locality with respect to the population grid and the search space; neighbors on the grid may represent solutions that are quite different. Two

Figure 3: A two dimensional grid used by GSGAs to define population subsets.

methods have been used to determine these subsets of solutions in GSGAs: (1) fixed size neighborhoods have been used to define the set of individuals about a given point on the grid, (2) random walks have been used to stochastically sample neighboring locations on the grid. For example, the dashed lines adjacent to point A in Figure 3 indicate a subset of individuals that might compete to replace the individual at point A. This subset is called the *neighborhood* of point A. The dashed lines adjacent to point B illustrate the toroidal nature of neighborhoods on this grid. We call a neighborhood like that shown in Figure 3 a NEWS neighborhood, since it only uses neighbors to the North, East, West and South.

Recently, Hart [10] has described a coarse grain design for parallel GSGAs. Like SIMD GSGAs, this parallel GA uses a two-dimensional toroidal grid that is distributed across the set of available processors. Thus each processor typically processes a set of solutions that is located on a subgrid of the entire grid. Communication of grid elements is performed to update the neighborhoods of individuals whose neighborhoods span two (or more) processors.

Figure 4: Block decomposition: (a) illustration and (b) border regions used to perform communication; darkly shaded regions are used for non-NEWS neighborhoods.

3 Methods

This section describes methods used in our experiments. The first subsection reviews the design of the coarse-grain parallel GSGAs that are described by Hart [10]. The second subsection details the parameters used in our experiments.

3.1 Parallel Design of Coarse-Grain GSGAs

Coarse grain GSGAs use a toroidal, two-dimensional population grid that is distributed across p processors. We examine GSGAs using HPF-style block decompositions [2]. Figure 4a illustrates this decomposition.

Communication between processors is required to (1) perform selection and recombination, and (2) check for termination signals. Because the grid is toroidal, periodic boundary conditions are required. Hence, every processor has the same number of neighbors to communicate with. Each processor may terminate independently by satisfying a termination criterion (e.g., by exceeding a specified time limit or finding a solution with a specified fit-

ness). Communication is required to terminate all of the processors when any of them satisfy the termination criterion.

Performing selection and recombination at a given location on the grid requires access to the fitness and genotypes of neighboring individuals, which may be located on other processors. Fixed size neighborhoods are used in our coarse-grain GSGAs, which lets us predetermine the size of the *border* regions that need to be communicated between processors. The number of of border regions that need to be communicated depends on the structure of the neighborhoods (see Figure 4b).

Coarse-Grain GSGAs can be distinguished by the manner in which interprocess communication is performed: (globally) synchronized or asynchronous. Synchronized GSGAs guarantee that all border regions for all of the processors have been communicated before any of the processors can proceed in the next iteration. By comparison, the asynchronous algorithm does not require that all border regions be filled prior to starting the next iteration [3]. Asynchronous GSGAs will probably have a higher utilization of the parallel hardware, but processors may frequently be using border regions that are inconsistent with the state of the neighboring processors.

3.2 Experimental Design

The GSGAs used in our experiments are similar to the those described by Hart [10]. Floating point encoding was used to represent solutions in $\mathbf{R}^n$. Mutation was performed by adding a value from a normally distributed random variable to a coordinate of a solution. The *minimal* neighborhood was used, which only includes the two immediate neighbors along

each dimension of the population grid (see Figure 3). The crossover rate was 0.8 and the mutation rate was 0.01. Rank selection was used to competitively select individuals within each neighborhood. Following Gordon and Whitley [8], we use the best individual in the neighborhood as one of the parents when performing crossover. Additional checks were added to avoid repeating a function evaluation on individuals that were not modified by the crossover or mutation operations.

Communication in our parallel GSGAs was coded using LPARX [12]. LPARX provides distributed 2D grid classes that handle block copies and global synchronization. LPARX was extended to create labeled block copies that are used to perform asynchronous communication.

The function that is optimized in our experiments is the two-dimensional molecular conformation problem described by Judson et al. [11]. This problem concerns a molecule composed of a chain of identical atoms that are connected with rigid rods of length one. The energy of the molecule is modeled van der Waals forces. The energy equation used by Judson et al. is $f(\alpha) = \sum_{i=0}^{n-1} \sum_{j=i+1}^n \left[\left(\frac{1}{r_{ij}} \right)^{12} - 2 \left(\frac{1}{r_{ij}} \right)^6 \right]$, where $r_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$. The angles α represent the bond angles for a molecule, which can be used to calculate the coordinates x_i and y_i (see Hart [10]). A molecule with 37 atoms was used in our experiments, which requires 35 bond angles to specify its conformational space. The optimal conformation has an energy of approximately -98.3.

The GSGAs were run on Sandia National Laboratories' 1824 node Intel Paragon under the SUNMOS operating system. Our reported results are averages over 20 trials with different seeds for the random number generator. Unless otherwise specified, all experiments were terminated when a solution with energy -70.0 was found.

4 Performance Analysis

4.1 Observations

There are several properties of parallel GAs that make the evaluation of their performance particularly challenging. These are discussed in the following paragraphs.

Randomization GAs are randomized algorithms that rely on random number generators to determine steps that they make. While serial algorithms use a single random number generator, it is impractical for parallel GAs to mimic the exact algorithmic sequence used by the sequential algorithm. Consequently, each processor in a parallel GA uses a random number generator with a distinct seed.

Thus the results of a parallel GA for two different sets of processors are not strictly comparable because the number of processors affects the sequence of algorithmic steps used by the algorithm. However, we argue that the *expected* performance of synchronized parallel GAs should not be affected. The expected behavior of synchronous parallel GAs does not depend upon the random number generators themselves, but on the distribution of search behaviors that they induce. This distribution is independent of the way in which the randomization is provided. It also seems reasonable that asynchronous parallel GAs are not affected by the different random number generators, but in this case the delays in inter-processor communication may influence the expected performance of parallel GAs in subtle ways.

Termination Criteria A weakness of GAs is the lack of a stopping criteria that can reliably terminate the algorithm near a local or global optimum. Consequently, researchers

typically terminate GAs after a fixed number of iterations or when the distribution of solutions in the population appears to have converged near a single point (total convergence typically never occurs due to the effects of mutation). These rules terminate GAs with solutions whose fitnesses vary. Consequently, researchers typically report the mean and standard deviation of the fitnesses of the best individuals found.

Performance measures for parallel algorithms are typically defined in terms of the cost required to find a solution with a given accuracy. Thus the performance analysis typically performed for GAs is inconsistent with the performance measures used for parallel algorithms. To analyze the performance of parallel GAs it is necessary to run the algorithm until a given accuracy is achieved since this is the only way of comparing results when the number of iteration varies with the number of processors.

Synchronization An interesting feature of many parallel GAs is that strictly speaking, synchronization is not required. The reason is that the search performed by each processor can proceed independent of the searches performed by the other processors. The role of communication is simply to prevent processors from searching "bad" parts of the space for too long. A processor that is searching a bad part of the space will eventually receive individuals from other processors that are relatively superior, thereby leading the processor to begin searching in the neighborhood of the new individuals. Consequently, asynchronous parallel GAs work well. This implies that a good performance measure for parallel GAs should be able to compare the performance of asynchronous and synchronous algorithms.

4.2 Performance Measures

The selection of a performance measure for parallel GAs is clearly influenced by the previous observations. For example, load imbalance is not a useful measure because it is not applicable to asynchronous parallel GAs.

The so-called “grind” measures also appear to have limited use for parallel GAs. Grind measures take the ratio between the total time used by a parallel algorithm and the number of iterations (or some other convenient algorithmic unit). The assumption made by this measure is that the cost of the algorithm is uniformly distributed across all iterations. In parallel GAs this is not the case. Figure 5a graphs the time per iteration for a typical run of a MIMD GSGA. The time per iteration clearly decreases as the algorithm proceeds. This can be attributed to the fact that the number of function evaluations decreases as the population becomes more homogeneous and crossover is more frequently applied to nearly identical solutions (see Figure 5b).

Now consider the well-known speedup measure. Let T_1 be the run time for the *best* sequential GA, and let T_p be the run time for p processors from the start of execution until the last processor has terminated. The *speedup* of a parallel algorithm is $\sigma_p = T_1/T_p$. The randomization of parallel GAs complicates the calculation of these performance measures because the sequence of algorithmic steps varies between a sequential implementation of the algorithm and its parallel implementation. Further, asynchronous GAs may have better numerical efficiency than synchronous GAs (as our experiments detail), so the value of T_1 is difficult to determine. Thus it is difficult to apply the speedup measure to parallel GAs.

Although the the speedup measure is not immediately applicable, the run time T_p can

Figure 5: A typical run of a MIMD GSGA (16 processors on a 160x160 grid) showing the expected (a) time per iteration and (b) number of function evaluations per iteration. Error bars show the standard deviation.

be used to provide a relative comparison between the speedups of two algorithms. If σ_p^1 and σ_p^2 are the speedups for two algorithms, then $\sigma_p^1 < \sigma_p^2$ if and only if $T_p^1 > T_p^2$. Thus T_p is the ultimate measure of performance for these algorithms. The relative parallel and numerical efficiencies of these algorithms must be independently determined.

5 Experimental Analysis

Our experimental analysis examines the numerical and parallel efficiency for parallel GSGAs. In particular, we examine the effect that synchronization has on both parallel and numerical efficiency. Our experiments compare the performance of synchronous and asynchronous coarse-grain GSGAs using a 160x160 toroidal population grid.

Figure 6 shows the expected run time, $E[T_p]$, for the parallel GSGAs with the population

Figure 6: Performance of synchronous and asynchronous coarse-grain GSGAs: (a) plot of T_p and (b) plot of pT_p .

grid subdivided among 4, 16, 64 and 256 processors. A comparison of the two curves in Figures 6a shows that the asynchronous GSGAs terminate more quickly than synchronous GSGAs, and consequently have a higher speedup. Since the population size remains fixed as the number of processors changes, these results may be comparable for different processor sizes if our conjecture concerning the effect of randomization in Section 4.1 is true.

To test this conjecture we compared the distribution of the total number of function evaluations used on all processors for each of the processor sizes. These distributions were compared using the method of multiple comparisons performed with the GH procedure [18]. This method tests multiple null hypotheses which state that the means of each pair of samples are identical, and the confidence of this test applies to all of the hypothesis tests considered collectively. This test indicated that the probability that any of the distributions can be distinguished is less than 5%. This confirms our conjecture.

Figure 7: Two factors affecting the relative run times of synchronous and asynchronous GSGAs: (a) communication overhead and (b) comparison of the total number of function evaluations used during optimization across all processors.

Figure 6b shows that as the number of processors increases, the run time decreases almost linearly for the asynchronous GSGAs and decreases sublinearly for synchronous GSGAs. In part, this difference can be attributed to the cost of synchronization. Figure 7a graphs the communication overhead, P_{comm} , the fraction of the total time spent in the communication subroutine. Note that for the asynchronous algorithm the time spent in the communication subroutine is a lower bound on the communication time, since communication can overlap the computation performed by the processors. Figure 7a illustrates the increased cost of synchronization, which is one factor that makes the run times for synchronous GSGAs higher than for asynchronous GSGAs.

The difference between the run times of asynchronous and synchronous GSGAs can also be attributed to differences in the numerical efficiency of the synchronous and asynchronous

algorithms. Figure 7b graphs the total number of function evaluations used by all processors during the optimization runs. The total number of function evaluations is a reasonable metric for evaluating numerical efficiency in our experiments because the function evaluations account for 70-90% of the total time across all processors.

The asynchronous GSGAs use fewer function evaluations, which indicates that asynchronous GSGAs actually optimize faster than synchronous GSGAs. The relative improvement is $(T_p^{sync} - T_p^{async})/T_p^{sync}$. We can evaluate the amount of relative improvement due to the reduction in function evaluations as follows. Let λ equal the fraction of function evaluations used in the synchronous algorithm that are used by the asynchronous algorithm, and let γ equal the fraction of T_p^{sync} that function evaluations account for. The relative improvement due to the reduction of function evaluations is then $(1 - \lambda)\gamma$. Figure 8 compares the relative improvement due to the reduction in function evaluations against the total relative improvement between the synchronous and asynchronous GSGAs. These results show that the reduction in the number of function evaluations accounts for a substantial fraction of the relative improvement between the synchronous and asynchronous GSGAs.

One possible explanation of the improved performance of the asynchronous GSGAs is that the delayed updates to the border regions caused by the asynchronous communication actually improve the search performed by the parallel algorithm. Intuitively, delayed communication allows each processor to search longer with its current population before becoming influenced by the solutions communicated by neighboring processors. To evaluate this hypothesis, we examined synchronous GSGAs that were synchronized at different frequencies with 256 processors. Table 1 shows the results of these experiments.

These results show that infrequent synchronization decreases the run time of coarse-grain

Figure 8: Relative improvement between the synchronous and asynchronous GSGAs.

Frequency	1	2	4	8	16
T_p	28.7	12.3	10.2	10.5	11.5
P_{comm}	0.261	0.217	0.194	0.191	0.194
Num Eval	8.25e5	3.66e5	3.07e5	3.18e5	3.46e5

Table 1: Effects of the frequency of synchronization on the performance of synchronous GSGAs.

GSGAs over fully synchronized GSGAs. This improvement can be attributed to improvements in both the parallel and numerical efficiency of the algorithm. The decrease in communication overhead is a natural consequence of a decrease in the synchronization frequency, though it is interesting that the decrease in communication overhead appears to level off. In part, this may be due to the fact that after 8 iterations, some of the trials have already terminated. The total number of function evaluations exhibit a sharp decline as the synchronization frequency is increased. These results are consistent with research on Island

	Accuracy	$E[P_{comm}]$	$E[T_p]$	StdDev T_p
Sync	-70	0.261	28.7	1.76
	-80	0.295	55.9	4.83
Async	-70	0.071	16.3	2.22
	-80	0.049	26.2	3.26

Table 2: Effects of performance accuracy on communication overhead and run time.

Model GAs (IMGAs) [17], which decompose the population into subpopulations, each of which is processed panmictically. The results confirm our hypothesis that delays in communication can improve the performance of parallel GAs, and they provide an explanation for the superior numerical efficiency of asynchronous GSGAs.

Finally, we consider how the performance of GSGAs is affected by the accuracy threshold. Table 2 shows the run time and communication overhead of GSGAs with 256 processors. The communication overhead is nearly constant as the accuracy threshold varies. The run time increases as the accuracy is tightened, which reflects the increased difficulty of the optimization problem. Furthermore, the standard deviation of the run time also increases. We interpret this to mean that the reliability of the algorithm's ability to quickly find a near-optimal solution decreases as the accuracy is tightened.

Figure 9 plots the standard deviation of T_p as the number of processors varies. The curves for the tighter accuracy are generally higher than the curves for the looser accuracy. However, the relative behavior between the synchronous and asynchronous GAs appears to be affected

by the accuracy tolerance. For the loose tolerance the asynchronous GA uniformly has a higher standard deviation. For the tight tolerance, the asynchronous GA only has a higher standard deviation when the number of processors is small.

Figure 9: The standard deviation of the run time, T_p , as the number of processors p varies.

6 Discussion and Conclusions

Our empirical results show that the numerical behavior of parallel GAs can be improved by altering the way that communication is implemented. This effect is independent of any reductions in processor idle time due to reduced communication overhead or load imbalance. We have examined two related strategies for altering interprocessor communication: (1) asynchronously dependent updates and (2) synchronous updates with reduced frequency. Both of these strategies delay the updates of border regions. Our experiments show that this type of delay improves the rate of convergence of the parallel GAs.

Because the performance of parallel GAs is sensitive to the way that communication is

implemented, traditional measures of parallel efficiency are not particularly meaningful for these algorithms. Consequently, our analysis has used the run time to distinguish the relative performance of algorithms, along with measurements that distinguish improvements in the communication overhead and the numerical performance. In this study we used the number of functions as a measure of the relative numerical performance of GAs. This is a standard metric for GAs, and it is a reasonable measure of the numerical performance when the total cost of function evaluations dominates the total cost of the computation.

The analysis of parallel GAs is complicated by several factors. Because GAs are probabilistic, a single trial of the algorithm is not particularly meaningful. Instead, the expected run time must be computed. Further, the implementation of randomization may vary with the number of processors, though our analysis indicates that this is not a factor that affects the expected performance of these algorithms. Another factor is the fact that GAs are only probabilistically guaranteed to terminate, and the run times may be highly variable for a given problem. This variability may prevent the performance analysis for very tight accuracy thresholds, since prohibitively long trials might be required to find near-optimal solutions. In our experiments the relative performance of the parallel GSGAs was consistent at different accuracy thresholds, which suggests that performance comparisons based on measurements with loose accuracy thresholds may apply for tighter accuracy thresholds.

Given our critique of traditional parallel performance measures, we expect that previous analyses of parallel GAs will need to be reconsidered. For example, several authors have used speedup measures to show that parallel GAs exhibit superlinear speedup [1, 14, 15]. Mühlenbein et al. [14] fix the number of individuals processed by a parallel IMGA and compare the performance for different numbers of processors. The problem with this comparison

is that the algorithmic structure is fundamentally changed as the number of processors changes. IMGAs perform independent panmictic searches on each processor. Thus there is no direct relationship between the computation performed on p processors and the computation performed on p' processors. Consequently, the meaning of this speedup measure is not clear.

Our experiments examine a single implementation issue for parallel GAs: the effect of synchronization on the parallel and numerical efficiencies of parallel GAs. There are a variety of other implementation issues whose effects need to be carefully examined. These include the topology of the population grid, the size of the population, and the type of neighborhoods used to perform competitive selection. Empirical evidence indicates that these issues can affect the numerical performance of GAs. For example, larger population sizes can sometimes improve the rate of convergence of GAs, particularly for difficult optimization problems.

While many of these issues have been examined by researchers, a careful analysis of their interactions in parallel GAs has not been conducted. Although we do not expect that a single formulation of parallel GAs will be best for solving all problems, this analysis is important for understanding what types of parallel GAs are likely to perform well. For example, our work in progress indicates that the shape of the population grid interacts with the synchronization method. Narrow grids appear to have better numerical performance, especially for asynchronous parallel GAs.

Acknowledgements

Scott Baden and Scott Kohn were supported by ONR contract N00014-93-1-0152. A UCSD School of Engineering Block Grant provided development time for the Intel Paragon located at the San Diego Supercomputer Center. Bill Hart was supported by the Applied Mathematical Sciences program, U.S. Department of Energy, Office of Energy Research, and this work was partially performed at Sandia National Laboratories, operated for the U.S. Department of Energy under contract No. DE-AC04-94AL85000.

References

- [1] T. C. Belding. The distributed genetic algorithm revisited. In L. Eshelman, editor, *Proceedings of the Sixth Intl. Conf. on Genetic Algorithms*, pages 114–121, San Mateo, CA, 1995. Morgan-Kaufmann.
- [2] B. Chapman, P. Mehrotra, and H. Zima. Extending HPF for advanced data parallel applications. Report 94-34, ICASE, May 1994.
- [3] D. Chazan and W. L. Miranker. Chaotic relaxation. *Lin. Algebra and Applic*, 2:199–222, 1969.
- [4] R. J. Collins and D. R. Jefferson. Selection in massively parallel genetic algorithms. In *Proceedings of the 4th International Conference on Genetic Algorithms*, pages 249–256, 1991.
- [5] L. Davis. *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, 1991.

- [6] L. J. Eshelman and J. D. Schaffer. Real-coded genetic algorithms and interval schemata. In L. D. Whitley, editor, *Foundations of Genetic Algorithms 2*, pages 187–202. Morgan-Kaufmann, San Mateo, CA, 1993.
- [7] D. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Publishing Co., Inc., 1989.
- [8] V. S. Gordon and D. Whitley. Serial and parallel genetic algorithms as function optimizers. In S. Forrest, editor, *Proceedings of the Fifth Intl. Conf. on Genetic Algorithms*, pages 177–183, San Mateo, CA, 1993. Morgan-Kaufmann.
- [9] M. Gorges-Schleuter. Explicit parallelism of genetic algorithms through population structures. In *Parallel Problem Solving from Nature*, pages 150–159, 1990.
- [10] W. E. Hart. *Adaptive Global Optimization with Local Search*. PhD thesis, University of California, San Diego, May 1994.
- [11] R. Judson, M. Colvin, J. Meza, A. Huffer, and D. Gutierrez. Do intelligent configuration search techniques outperform random search for large molecules? *International Journal of Quantum Chemistry*, pages 277–290, 1992.
- [12] S. R. Kohn and S. B. Baden. A robust parallel programming model for dynamic non-uniform scientific computations. In *Proceedings of the 1994 Scalable High Performance Computing Conference*, May 1994.
- [13] J. M. McInerney. *Biologically Influenced Algorithms and Parallelism in Non-Linear Optimization*. PhD thesis, University of California, San Diego, 1992.

- [14] H. Mühlenbein, M. Schomisch, and J. Born. The parallel genetic algorithm as function optimizer. In R. K. Belew and L. B. Booker, editors, *Proceedings of the Fourth Intl. Conf. on Genetic Algorithms*, pages 271–278, San Mateo, CA, 1991. Morgan-Kaufmann.
- [15] R. Shonkwiler. Parallel genetic algorithms. In S. Forrest, editor, *Proceedings of the Fifth Intl. Conf. on Genetic Algorithms*, pages 199–205, San Mateo, CA, 1993. Morgan-Kaufmann.
- [16] P. Spiessens and B. Manderick. A massively parallel genetic algorithm: Implementation and first analysis. In *Proceedings of the 4th International Conference on Genetic Algorithms*, pages 279–285, 1991.
- [17] T. Starkweather, D. Whitley, and K. Mathias. Optimization using distributed genetic algorithms. In *Parallel Problem Solving from Nature*, pages 176–185, 1990.
- [18] L. E. Toothaker. *Multiple Comparisons for Researchers*. Sages Publications, 1991.

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.