Parallelization of Elimination Tournament Selection without Synchronization

Marin Golub, Domagoj Jakobović, Leo Budin
Department of Electronics, Microelectronics, Computer and Intelligent Systems
Faculty of Electrical Engineering and Computing
Unska 3, HR-10000 Zagreb, Croatia
e-mail: {marin.golub, domagoj.jakobovic, leo.budin}@fer.hr

Abstract — A global parallel genetic algorithm (GPGA) with elimination tournament selection without duplicates is described in this paper. The parallel implementation of genetic algorithm with multiple threads without synchronization is thoroughly investigated. The problem which occurs when several threads select the same individual for elimination is recognized in such implementation. The probability of that event is analytically formulated. Considering that probability, we determine the total number of iterations the asynchronous GPGA should perform in order to achieve the same optimization effect as a serial GA or a synchronous GPGA.

I. INTRODUCTION

It has already been shown that the tournament selection is more suitable for parallel execution than any other selection method. Furthermore, the tournament selection algorithm is rather straightforward and relatively simple to implement. It requires only a very small amount of computation time [7,8,10,16].

The selection operator in genetic algorithm is responsible for promoting the good individuals in the population and eliminating the bad ones. Various selection methods differ only in the definition of 'good' and 'bad' individuals and in the ways of their preserving or elimination. A good selection choice may be the only difference between poor and (at least) acceptable genetic algorithm efficiency in many real life applications.

The tournament selection bears one parameter usually denoted with *k*. The *k*-tournament selection randomly chooses *k* individuals with equal probability from the population. Then, depending of the selection type, generational *k*-tournament selection copies the best individual of the *k* selected ones in the mating pool. The process is repeated until *N* individuals are copied in the mating pool, where *N* is the size of the population. After that, the reproduction operators are applied on the mating pool to complete the creation of a new generation. Elimination *k*-tournament selection, on the other hand, eliminates the weakest one of the *k* selected individuals. Each eliminated individual is replaced with a newly generated one using the reproduction operator.

The generational selection type has two significant drawbacks: it creates duplicates of better individuals and places them in the mating pool. Duplicates in the mating pool will slow down the whole optimization process. Furthermore, they can lead the population to a local optimum and we have to rely on the reproduction operator alone to maintain the population diversity. The second drawback lies in the fact that the reproduction operator has to be applied after the selection is finished. This implies the sequential nature of the algorithm and calls for some sort of synchronization mechanism. Such a mechanism may slow down the parallel algorithm and it also makes the implementation somewhat more complicated and less portable [4]. The elimination selection type avoids both mentioned problems: it does not generate any duplicates; it only eliminates a single population member. The eliminated individual can be immediately replaced by reproduction. Furthermore, the mechanism of preservation of the best individual, also called elitism, is implicitly included in the elimination selection, as it will be shown in following sections. That is another reason to implement this selection type in a genetic algorithm.

The parallel genetic algorithm presented in this work is an instant of global parallel GA (GPGA). The attribute 'global' refers to an algorithm that maintains a single population. More than one genetic operator, implemented in a single process or a thread, can access the population in the same time. Global also means, since we have one unique population, that any individual can compete for survival or mate and produce offspring with any other individual from the population. Such a model of PGA is also called master-slave genetic algorithm [5,6].

The GPGA described in this work is suitable for executing on a multiprocessor system with shared memory and operating system which supports multithreading. The multithreading technique is recognized as an efficient tool for transforming the genetic algorithm into parallel form. The maximum efficiency is achieved when the number of threads is equal to the number of processors.

II. THE SELECTION PROBABILITY

Let the population consist of *N* individuals and let the individuals are indexed by their fitness value, i.e. the best one has the index *i*=1 and the worst one *i*=*N*. The selection probability for the *k*-tournament selection is given by:

$$p_k(i) = \frac{\binom{i}{k} - \binom{i-1}{k}}{\binom{N}{k}}. \qquad (1)$$

The *i*-th individual will be selected for elimination if all the other *k*-1 chosen population members are better than the *i*-th. In other words, all but one of *k* selected individuals' indexes are less than *i* and one individual has index i. Now, the probability that all k randomly selected individuals have their indexes *less or equal* than i is $\binom{i}{k}$ divided by all possible combinations: $\binom{N}{k}$.

From that we have to subtract the probability that all of the k individuals' indexes are strictly less than i: $\binom{i-1}{k}$ divided by $\binom{N}{k}$,(i-th individual is selected, and the rest ones' indexes are less than i), so we get the mentioned expression (1).

Taking into account the following statement:

$$\binom{n}{k} = \frac{n \cdot (n-1) \cdot (n-2) \cdot \ldots \cdot (n-k+1)}{k!} . \quad (2)$$

another form of (1) can be derived:

$$p_k(i) = \frac{k}{N} \prod_{j=1}^{k-1} \frac{(i-j)}{(N-j)} . \quad (3)$$

The elitism is inherently implemented, because the k-1 best individuals can not be eliminated. The expression (3), being a polynomial with k-1 zero crossings, illustrates that fact.

The same individual cannot participate in a tournament more than once, i.e. there cannot be more that one copy of the same individual among the k selected. This we call the tournament selection without duplicates. On the other hand, Bäck, Miller, Goldberg, Blickle and Thiele use tournament selection where in the same iteration one individual can be selected more then once [1,3,13]. In that case, the probability $p_k'(i)$ of selection for elimination of the i-th individual is given by:

$$p_k'(i) = \left(\frac{i}{N}\right)^k - \left(\frac{i-1}{N}\right)^k . \quad (4)$$

At the first sight, it would seem that tournament selection without duplicates is more complex and time consuming. Besides, the characteristic values such as selection intensity, reproduction rate and loss of diversity have already been determined for duplicate tournament selection [1,13]. Genetic algorithm's behavior can be fairly predicted using those values. However, if no control mechanism is applied, elimination tournament selection can produce additional duplicates, which only slow down the evolution process. Such control mechanism, an example of which can be found in [12], eliminates duplicate individuals, but spends much more computation time then relatively simple duplicate check in a single tournament. Furthermore, the elitism is not inherently implemented, unlike in no-duplicates tournament selection, which can be perceived from (3) if i equals 1. Although the expressions for selection probability differ significantly, the actual distributions are very similar, as shown in Fig. 1 [9,11].

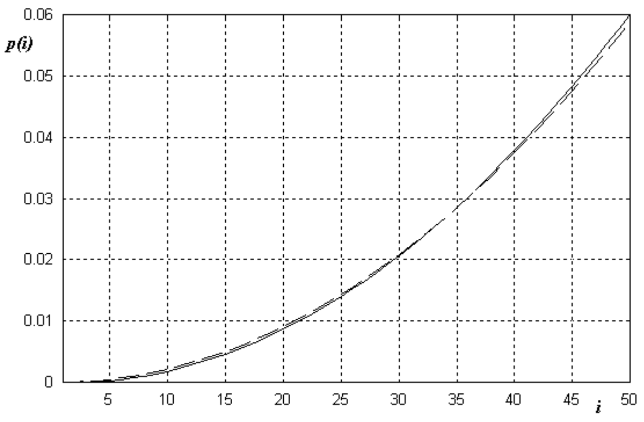

Fig. 1: Distribution of selection for elimination probability with 3-tournament selection and N=50 with (_ _ _) and without (____) duplicates

III. EXAMPLE OF GPGA WITH ELIMINATION TOURNAMENT SELECTION

In the traditional master-slave model of the parallel GA, the master processor stores the entire population and applies genetic operators to produce the next generation [2,3,7,8,13]. The slave processors are used only to evaluate the fitness of a fraction of the population in parallel. This type of task division evolves from a situation where the calculation of the fitness value of a population member requires a lot of computation, if the potential solution to a problem we are trying to solve is not easy to evaluate. In our implementation of master-slave GA, the master creates random initial population, evaluates created individuals and starts the slaves (Fig. 2). Each slave performs the whole evolution process in contrast of traditional master-slave GA where the slaves only evaluate the fitness. This is a model of global parallel GA, because each individual may compete and mate with any other [9,10].

```
Master thread{
 initialize population;
 evaluate population;
 for(i=1;i<NUMBER_OF_PROCESSORS;i++){
   create new Slave thread;
 }
 wait while all threads finish;
 print results;
}

Slave thread{
 while not(end condition){
   select k different individuals;
   eliminate the worst of k selected ind;
   child=crossover(survived individuals);
   replace deleted individual with child;
   perform mutation with probability p_m;
   evaluate new individual;
 }
}
```

Fig. 2: ***Asynchronous GPGA with k-tournament bad individual selection***

In this implementation, which we call the asynchronous GPGA, it may occur that the same individual participates in more that one tournament, i.e. the same individual can be selected by more that one thread. To cope with that, the synchronous version of the algorithm is devised. However, the code of the synchronous GPGA is more complex and additional processor time is spent on initialization, locking and unlocking of MUTEX mechanism. Furthermore, a thread may have to wait for another one to complete its task.

In an asynchronous version, the selection of the same individual in more than one tournament is still not the reason for alarm. The worst case scenario happens when several threads select a single individual for *elimination*. In that case the work of only one of them will take effect – the one that replaced the individual latest – while the other threads will work in vain. The total number of iterations (tournament cycles) does not reflect the effective number of iterations because some of them had no effect. That is the reason why asynchronous algorithm does not give as good solution as the same sequential GA in given number of iterations. But if we know the probability of multiple elimination of the same individual by several threads at the same time, we can calculate the number of iterations done in vain. If the total number of iterations is increased according to that probability, the asynchronous GPGA will act exactly as, but faster than, sequential GA.

IV. CALCULATING THE PROBABILITY OF MULTIPLE SELECTION

A. *The Probability of a Particular Thread Working in Vain in a Single Iteration*

Let the asynchronous GPGA consist of D threads:

$$j_1, j_2, j_3, \ldots j_D.$$

Let each thread in every iteration (every time it accesses shared data) bears an index in range 1,2,3,...,D and let the thread which writes the data last have the smallest index value. That is the only thread whose work will take effect. Since the algorithm is asynchronous, the threads access the data in random order. In every iteration there is a thread with index value 1 and in every iteration it can be a different thread. The thread with index value greater than 1 may, but doesn't necessarily have to, perform its task in vain.

For D threads and k-tournament selection we have the following:

- The thread with index 1 never works in vain:

$$P'_{k,1} = 0 . \qquad (5)$$

- For each additional thread the probability of working in vain equals 1 minus the probability of performing *useless* iteration (p=1-q). It will be shown later in this section that the probability of working in vain for thread with index j equals:

$$P'_{k,j} = 1 - \sum_{i=1}^{N} p_k(i) \cdot q_k(i)^{j-1} , \qquad (6)$$

where $j \leq D$ and $p+q=1$.

The probability of elimination of i-th individual in k-tournament selection is calculated using (1) or (3). The thread with index j, which selected some individual for elimination, will be effective only if all the threads with indexes smaller than j select some other individuals for elimination. The probability of selection of i-th individual by thread j equals $p_k(i) \cdot q_k(i)^{j-1}$, because j-1 threads (whose indexes are less than j) have to select some other individual instead of i-th one. There exist N combinations where a thread can work effectively: it can select any of the N individuals whereas the other threads must not select the same one. Having that in mind, we can formulate the probability of thread j not working in vain for one iteration as a sum of probabilities of single selection of each individual by that thread:

$$Q'_{k,j} = \sum_{i=1}^{N} p_k(i) \cdot q_k(i)^{j-1} . \qquad (7)$$

The probability that the thread's activity will *not* take effect is the opposite of the stated one ($P'_{k,j}$=1-$Q'_{k,j}$), so we get the expression (6).

B. *The Probability of Any Thread Working in Vain in a Single Iteration*

Considering the fact that every thread in asynchronous algorithm can have any index value with equal probability, the probability of a thread working in vain in k-tournament GPGA with D threads equals:

$$P_{k,D} = \frac{1}{D} \sum_{d=1}^{D} P'_{k,d} . \qquad (8)$$

Using (6) and (8) we can obtain the expression for probability of working in vain in a single iteration:

$$P_{k,D} = 1 - \frac{1}{D} \sum_{d=1}^{D} \sum_{i=1}^{N} p_k(i) \cdot q_k(i)^{d-1} , \qquad (9)$$

and the probability of performing a useful iteration equals:

$$Q_{k,D} = \frac{1}{D} \sum_{d=1}^{D} \sum_{i=1}^{N} p_k(i) \cdot q_k(i)^{d-1} . \qquad (10)$$

The values of expected probability of performing a useless iteration for 3-tournament selection are shown in Table I. Fig. 3 shows the same values as a function of the number of threads with a constant population size.

Suppose, for example, that we want to achieve efficiency of asynchronous GPGA to 95% (restrict the probability of performing a useless iteration is less than 5%). The minimum population size is the function of number of threads (grayed area in Table I). For a greater number of threads the population size should be increased as shown in (9) and in Fig. 3.

TABLE I
PROBABILITY OF PERFORMING A USELESS ITERATION DEPENDING ON THE POPULATION SIZE *N*, THE NUMBER OF THREADS *D* AND *k*=3

	Number of threads *D*								
N	1	2	3	4	5	6	8	10	20
10	0	0.100	0.185	0.257	0.318	0.371	0.456	0.522	0.704
20	0	0.047	0.091	0.132	0.170	0.205	0.267	0.322	0.512
50	0	0.018	0.036	0.054	0.070	0.087	0.118	0.148	0.273
70	0	0.013	0.026	0.038	0.051	0.063	0.086	0.108	0.208
100	0	0.009	0.018	0.027	0.036	0.044	0.061	0.077	0.152
150	0	0.006	0.012	0.018	0.024	0.030	0.041	0.052	0.106
200	0	0.005	0.009	0.014	0.018	0.022	0.031	0.040	0.081
500	0	0.002	0.004	0.005	0.007	0.009	0.013	0.016	0.033

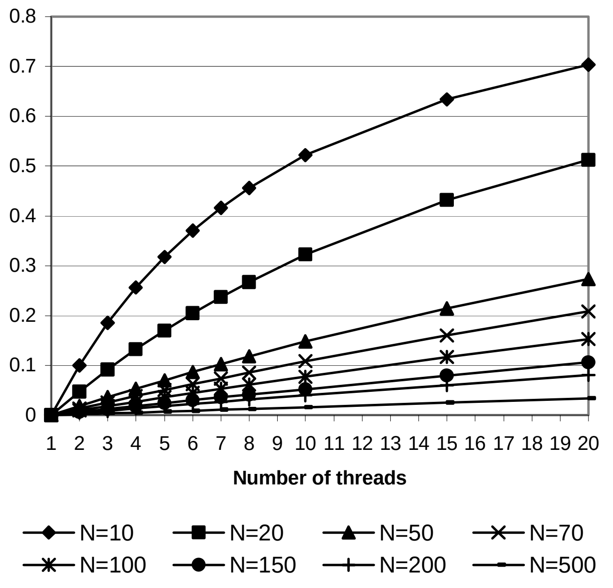

Fig. 3: Probability of performing useless iterations

C. *The Total Number of Iterations*

Knowing the probability of a thread working effectively in a single iteration (10), we can calculate the needed number of iterations that asynchronous GPGA with *k*-tournament selections must perform in order to have the same characteristics as a synchronous GPGA or serial GA [10]:

$$I_T = \frac{I}{Q_{k,D}}. \qquad (11)$$

V. CONCLUDING REMARKS

We presented in this paper a variant of a global parallel GA without synchronization mechanism. The asynchronous GPGA is faster and simpler to implement than the synchronous version. However, for a given number of iterations the synchronous GPGA gives slightly better results. This is due to the occurrence of multiple selection of the same individual by several threads in asynchronous version, in which case some iterations of the algorithm will be lost. It is shown in this work that we can explicitly calculate that overhead for a given number of threads and population size.

The probability of multiple selection increases with the number of processors and decreases with the population size (Fig. 3). The real world problems usually require relatively large population sizes. For example, if the population size is greater than 100 and we have a multiprocessor system with four processors and shared memory (in which the whole population is placed), the percentage of useless iterations is less than three percent (Table I). We can conclude that for GPGA implementations with large population sizes, running on multiprocessor systems with several processors, the asynchronous version is better choice.

ACKNOWLEDGEMENTS

This work was carried out within the projects: 036-014 *Problem-Solving Environments in Engineering* and 036-814 *Complex Functions Optimizing Using Evolution Programs*, funded by Ministry of Science and Technology of the Republic of Croatia.

REFERENCES

[1] Bäck, T., "Selective Pressure in Evolutionary Algorithms: A Characterization of Selection Mechanisms", Proc. of the First IEEE Conference on Evolutionary Computation, IEEE Press, Piscataway NJ, pp. 57-62, 1994., available from: http://ls11-www.informatik.uni-dortmund.de/people/baeck/papers/wcci94-sel.ps.gz.

[2] Bäck, T., "Generalized Convergence Models for Tournament- and (μ,λ)-Selection", Proceedings of the Sixth International Conference on Genetic Algorithms, San Francisco, CA, pp. 2-8, 1995.

[3] Blickle, T., Thiele, L., "A Mathematical Analysis of Tournament Selection", Proc. of the Sixth International Conference on Genetic Algorithms, San Francisco, CA, pp. 2-8, 1995.

[4] Budin, L., Golub, M., Jakobović, D., "Parallel Adaptive Genetic Algorithm", International ICSC/IFAC Symposium on Neural Computation NC'98, Vienna, 1998, pp. 157-163.

[5] Cantú-Paz E., "A Summary of Research on Parallel Genetic Algorithms", 1995., available from: www.dai.ed.ac.uk/groups/evalg/Local_Copies_of_Papers/Cantu-Paz.A_Summary_of_Research_on_Parallel_Genetic_Algorithms.ps.gz

[6] Cantú-Paz, E., "A Survey of Parallel Genetic Algorithms", Calculateurs Paralleles, Vol. 10, No. 2. Paris: Hermes, 1998., available via ftp from: ftp://ftp-illigal.ge.uiuc.edu/pub/papers/Publications/cantupaz/survey.ps.Z.

[7] Cantú-Paz, E., Goldberg, D.E., "Parallel Genetic Algorithms with Distributed Panmictic Populations", 1999., available from: http://www-illigal.ge.uiuc.edu/cgi-bin/orderform/orderform.cgi.

[8] Cantú-Paz, E., "Migration Policies, Selection Pressure, and Parallel Evolutionary Algorithms", 1999., available via ftp from: ftp://ftp-illigal.ge.uiuc.edu/pub/papers/IlliGALs/99015.ps.Z.

[9] Golub, M., Jaković, D., "A New Model of Global Parallel Genetic Algorithm", Proceedings of the 22nd International Conference ITI2000, Pula, 2000, pp. 363-368.

[10] Golub, M., Budin, L., "An Asynchronous Model of Global Parallel Genetic Algorithms", Second ICSC Symposium on Engineering of Intelligent Systems EIS2000, University of Paisley, Scotland, UK, 2000, pp. 353-359.

[11] Golub, M., "Improving the Efficiency of Parallel Genetic Algorithms", Ph.D. Thesis, Zagreb, 2001. (in Croatian)

[12] Michalewicz, Z., "Genetic Algorithms + Data Structures = Evolutionary Programs", Springer-Verlag, Berlin, 1992.

[13] Miller, B.L., Goldberg, D.E., "GAs Tournament Selection and the Effects of Noise", 1995., available from: http://www.dai.ed.ac.uk/groups/evalg/Local_Copies_of_Papers/Miller.Goldberg.GAs_Tournament_Selection_and_the_Effects_of_Noise.ps.gz.

[14] Muehlenbein, H., "Evolution in Time and Space - The Parallel Genetic Algorithm", Foundations of Genetic Algorithms, G. Rawlins (ed.), pp. 316-337, Morgan-Kaufman, 1991., available via ftp from: ftp://borneo.gmd.de/pub/as/ga/gmd_as_ga-91_01.ps

[15] Munetomo, M., Takai, Y., Sato, Y., "An Efficient Migration Scheme for Subpopulation-Based Asynchronously PGA", Hokkaido University Information Engineering Technical Report HIER-IS-9301, Sapporo, July, 1993.

[16] Yoshida, N., Yasuoka, T., Moriki, T., "Parallel and Distributed Processing in VLSI Implementation of Genetic Algorithms", Proceedings of the Third International ICSC Symposia on Intelligent Industrial Autimation, IIA'99 and Soft Computnig, SOCO'99, June 1-4, Genova, Italy, pp. 450-454, 1999.