An Investigation of Topologies and Migration Schemes for Asynchronous Distributed Evolutionary Algorithms

Muhannad Hijaze
School of MACS, Heriot-Watt University
Edinburgh, UK
mh132@hw.ac.uk

David Corne
School of MACS, Heriot-Watt University
Edinburgh, UK
dwcorne@macs.hw.ac.uk

***Abstract*—Distributed evolutionary algorithms are of increasing interest and importance for three main reasons: (i) a well designed dEA can outperform a 'standard' EA in terms of reliability, solution quality, and speed; (ii) they can (of course) be implemented on parallel hardware, and hence combine efficient utilization of parallel resources with very fast and reliable optimization; (iii) parallel hardware resources are increasingly common. A dEA operates as separate evolving populations with occasional interaction between them via 'migration'. A specific dEA is characterized by the topology and nature of these interactions. Although the field is sizeable, there is still relatively little exploration of the performance of alternative topologies and interaction mechanisms. In this paper we compare some simple, novel dEA topologies with the cube-based topology that forms the basis of Alba et al's GD-RCGA (a state of the art dEA). We find the best results (when topologies are compared on a like for like basis in terms of number of processors) emerge from a three-level tree-based topology.**

Keywords- function optimization, evolutionary algorithms, parallel evolutionary algorithms.

I. INTRODUCTION

Distributed evolutionary algorithms (dEAs) operate by having several independent populations of chromosomes, with occasional interaction between them. In a typical dEA, the separate populations will evolve independently for a number of generations, and then "migration" will occur, in which chromosomes from one or more of the populations (or 'demes') will be copied into one or more of the other populations. The populations then continue to evolve independently until the next migration event, and so on. Such an EA design is well known to have several desirable properties. Not least is the fact that several dEAs are well known to perform more successfully than standard EAs that are otherwise the same (e.g. in terms of operators and total population size) – that is, *when implemented on a single processor*, both solution quality and speed (in terms of total fitness evaluations) can be very favourable. Secondly, dEAs are of course highly parallelizable, and offer a highly natural way to exploit a variety of different parallel architectures. The latter advantage is becoming increasingly more important as parallel hardware resources

There are two main kinds of dEA (note that these are often simply called 'parallel genetic algorithms). In the simplest, a standard EA is distributed over several processors but otherwise is little changed algorithmically. The alterations to the EA itself are only those necessary, if any, to enable exploitation of the parallel hardware. Examples include [1,2].

The second (and main) line of research in distributed EAs involves the establishment of (largely) independent (sub)populations, each using its own processor. In such a scheme, a 'migration' strategy is used to communicate information between processors at intervals [3]. There are a great variety of alternative migration schemes. The migration strategy is typically to copy good chromosomes from some populations to others. The broad dynamics of such a dEA amount to healthy forms of exploration (promoted by independent subpopulations, mostly non-interacting) and exploitation (promoted by migration) that are not shared by single-population EAs. As mentioned, this typically leads to improved performance in terms of both solution quality and speed.

Such dEAs are often termed either *fine-grained* or *coarse-grained*, depending on the sizes of the sub-populations. The 'ultimate' fine-grained models have a single individual per processor (e.g. [4]). A more favoured approach, which is more in tune with the majority of hardware configurations, is the coarse grained model (e.g. [5—9]. The Genitor algorithm [8] is a particularly well-known example, in which the subpopulations were linked in a ring topology – migration events involved chromsomes being copied from a population to its immediate neighbours in the ring. A more recent example is that of [7], in which the populations are linked in a cube topology, in which subpopulations are vertices of the cube, and each is linked to three others along the edges of the cube. We explore such a cube topology in this paper; the dGA model explored in [7] has several other sophistications; we do not implement those here, being interested for current purposes only in the performance of alternative simple topologies.

The remainder of the paper is set out as follows. Section II describes our basic parallel EA control strategy and migration/interaction mechanism, and introduces the topologies that we test in this paper. Section III then covers some experimental design details, including the test functions and the evaluation strategy. In section IV we present our results and an associated discussion. We provide a concluding discussion in Section IV.

II. Algorithms, Topologies and Interactions

A. EA Pseudocode: Master and Client Threads

Our dEA implementations are all physically parallel and asynchronous, utilizing a collection of standard workstations via sockets technology. In each case, the implementation is done via a master thread and several client threads.

The basic operation, in all models, is as follows. Following initialization, in which connections are established, a Master thread receives continual updates from each client thread concerning their best chromosomes and the associated fitness. When a chromosome is found that improves the best fitness so far, the master sends this to the client that currently has the worst "best fitness" according to its latest information. Meanwhile, client threads operate the population on a single processor, and incorporate new chromosomes as and when they are sent by the master thread. Whenever a new chromosome is received, if it is fitter than the current best in that population, then it is included in the population and the current worst is discarded. At frequent intervals, each client sends its best chromosome to the master thread.

The first model we discuss (model 1) operates in precisely the way above, and uses a straightforward architecture in which the master is directly connected to each client. In models 2 and 3, however, the topology is altered, and the master thread connects to a restricted number of clients, depending on the topology. In each case, the master connects to a group of clients, and the group of clients communicate directly with each other. We now present pseudocode clarifying the operation of the master and client threads. In the case of model 1, the pseudocode describes exactly what goes on. In the cases of models 2 and 3, there are differences that will be clarified later. In essence, however, the topology defines a set of (perhaps overlapping) groups of clients, and each client operates as both a master and a client within its group, while an overall master thread operates over the groups.

The overall responsibilities of the master thread are as follows:

1. Establish connections between the clients
2. Establish and initialize parameters
3. Start and Terminate the optimization
4. Receive and store up to date data from clients
5. Distribute appropriate data to clients

The master pseudo code operates three threads, as follows:

Thread1:

```
Connect Master with Clients;
For each Client:
   Send all parameters;
   Send "Start Process";
End for
Run Thread2; // for receiving data
Run Thread3; // for sending data
```

Thread2:

```
Repeat
   For each Client
     Listen to Connection Stream and store
     received data in DataString;
     Decode DataString and Convert it to
     real Chromosome and Fitness;
     If OptimChrom = Null // first one
          Set OptimChrom;
          Set OptimFitVal;
     Else if this Fitness is better
     than OptimFitVal
          Set OptimChromosome;
          Set OptimFitVal;
     End if;
  End for;
Until target Fitness Value reached, or max
time reached
```

Thread3:

```
Repeat
    If there is a new OptimChrom
        Send OptimChrom and its fitness
        back to worst client;
    Sleep(100);
Until Reach target Fitness Value;
```

The client threads operate as follows. Thread 1 is responsible for connecting to and receiving data from the master thread. It connects with the master, and waist for a "Start Process" signal. When this is received, it starts threads 2 and 3. Thread 2 runs the evolutionary algorithm on that client's processor, and thread 3 takes care of sending updated data on this client's best chromosome to the master.

Thread 1:

```
wait for "Start Process" signal from master;
Run Thread 2;
Run Thread 3;
While connection is established
       Store received data in DataString;
       Decode and Convert DataString to
           NewChrom and its FitnessValue;
       If new FitnessValue is better than
        Current OptimFitVal
              Set OptimChromosome;
              Set OptimFitVal;
              Replace worst chromosome in
              Population with NewChrom
       End if
End While
```

Thread 2:

```
Create and Initialise Population;
Set Count = 0;
Find best chromosome and call it OptimChrom;
Set OptFitVal;
Repeat
       Count++;
       Select chromosomes c1 and c2
       Crossover(c1,c2), producing children
         c1' and c2'
       Mutate(c1'), producing c1''
       Mutate(c2'), producing c2''
       Calculate fitness of c1'';
       Calculate fitness of c2'';
       If fitness(c1'') better than OptFitVal
              SendNewData=True;
              Set OptFitVal = fitness(c1'')
              Set OptChrom = c1''
       If fitness(c2'') better than OptFitVal
              SendNewData=True;
              Set OptFitVal = fitness(c1'')
              Set OptChrom = c1''
Until OptFitVal <= TargetFitVal or
  Count=MaxCount
```

Thread 3:

```
While Thread2 is running
       If SendNewData
              Send OptChrom and its fitness
               to Master;
              SendNewData=False;
              Sleep(100);
       End If;
End While;
```

B. Evolutionary Algorithm and Other Implementation Details

In all models we use a generational evolutionary algorithm that operates with a truncation selection strategy [10], in which, in each generation, the best 33% of the population are retained, and, selecting only from this best 33%, crossover and mutation are employed to generate the remaining 66% of the new population.

Our EA uses crossover and mutation operators detailed in [7]; specifically the fuzzy connective-based crossover operators: F- Crossover, S- Crossover, L- Crossover, M-Crossover, along with one-point, two-point, and uniform crossover. Each time a crossover operator is applied, it is a uniform random choice between these seven. Mutation (Gaussian mutation of a single randomly chosen parameter) is performed on a child of crossover with probability 0.25.

All our models use 15 individuals per subpopulation and (where applicable) a migration is performed every 25 generations. The probability of update an individual by mutation is 0.25, and the crossover probability is 0.6. There is a predefined maximum number of generations (10000), but a trial run will terminate if it has reached the target fitness values.

The physical hardware used is a cluster of eight personal computers running Microsoft widows XP Professional SP3, each one having an Intel Pentium IV 2.99 GHz processor and 2 GB of memory. The machines are interconnected by a Fast-Ethernet (100 Mbps) network.

C. Topologies

In model 1 (T1), the master connects directly to each of 16 clients, and clients operate two per processor (i.e. are distributed over 8 machines). Figure 1 illustrates model 1.

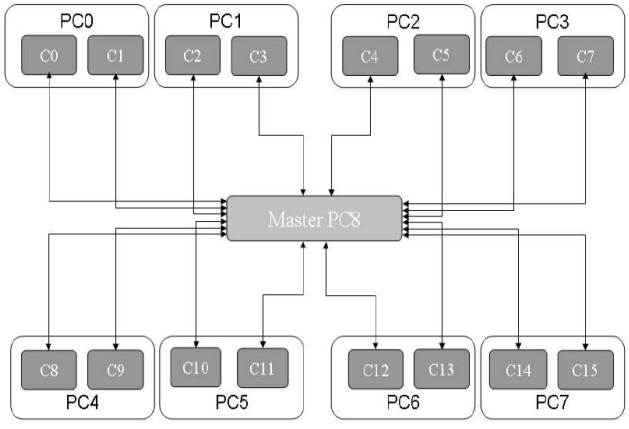

Figure 1. Model 1 topology for 16 clients.

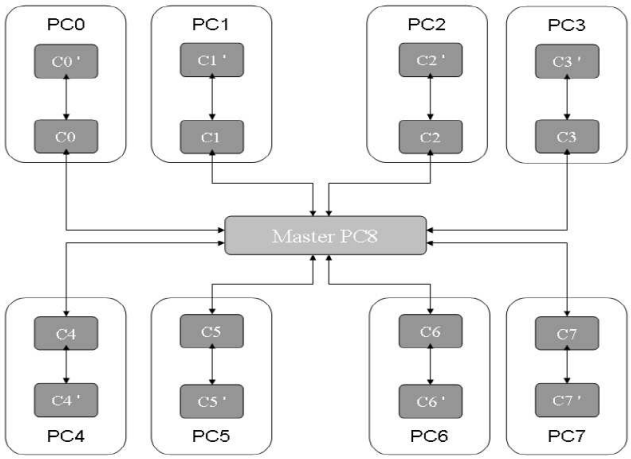

Figure 2. Model 2 topology for 16 clients.

Model 2 (T2) differs in a simple way from model 1, but in a way which leads to potentially quite different search dynamics. In Model 2, which is illustrated in Fig. 2., each master connects directly with each of eight groups, and each group consists of precisely two intercommunicating clients. Recall that, with the basic model (model 1), the master keeps track of the current overall optimum, and will copy this to the current *worst* client whenever a new optimum is discovered. This is so for model 2 at the level of groups; in model 2, a new best chromosome will first be transferred to the sister client in its group, and will soon appear in both clients of the previously worst group – hence there is, in some sense, more exploitation of newly discovered good chromosomes in model 2.

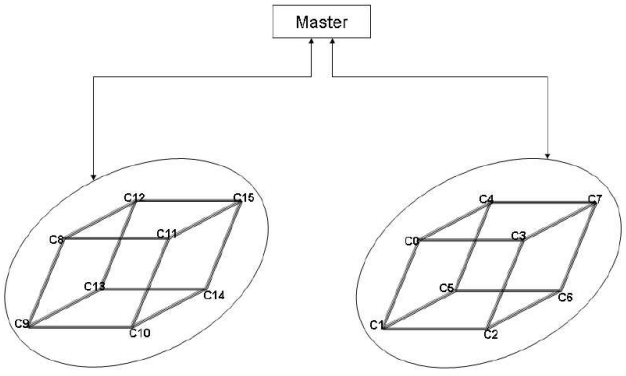

Figure 3. Model 3 topology for 16 clients.

In model 3 (T3), illustrated in Fig. 3., we base the topology and interactions on the model of Alba et al [7], in which a master controls two overall groups, but each group has a number of overlapping subgroups based on a cube topology. In this model, each client is connected directly to three other clients – the ones with which it shares an edge along the cube. When such a client finds a new best chromosome (with respect to only its own population) it chooses a random one of its neighbours, and sends the chromosome to that neighbour. Each client does this every 25 generations. At the master level, there are just two groups; every 25 generations, the best chromosome within the first group (i.e. from all populations in the first cube) is sent to a randomly chosen client from the second cube. The same then happens vice versa. Finally, we tested both 8-population and 16-population examples of each model. In the 8-population cases, the details of T1 and T2 (using 4 groups of 2) are straightforward. In the case of T3, this resulted in using just one cube, and hence did not involve migration at a higher level than the cube itself.

III. EXPERIMENTS

A. *Test Functions*

We compare the three models by using six well-known test functions, and use two versions of each – the standard version, and the 'shifted' version in which the global optimum is subject to random translation, rendering the function less decomposable. In Table I we indicate the functions used, and also indicate the target fitness. In most cases this is the global optimum. In all experiments, the functions had a dimension of 30 (i.e. 30 parameters).

TABLE I. FUNCTIONS USED IN EXPERIMENTS: TARGET FITNESSES

Function	Shifted	Not Shifted
Sphere	0	0
Rosenbrock	3.5	0
Schwefel	$5E10^{-7}$	0
Rastrigin	0	0
Griewangk	0	0
Ackley	0	0

We tested each model (T1, T2 and T3) by running 20 independent trials on each of these 12 test functions, and doing this for each of an 8-population and 16-population case. We first indicate the raw results in terms of success rates – i.e. the number of times that each set of 20 runs resulted in finding a globally optimal chromosome. Note that there is no particular prior expectation that this will be large (or even above 0) in many cases, however the dynamics of parallel search, in conjunction with the operators chosen from [7], can be highly effective.

TABLE II. SUCCESS RATES (OUT OF 20) FOR 8-POPULATION VERSIONS (1-CLIENT INDICATES SINGLE POPULATION SERIAL METHOD WITH EQUIVALENT OVERALL POPULATION SIZE): NON-SHIFTED AND SHIFTED FUNCTIONS

Function	1-client	T1	T2	T3
Sphere	18 (20)	19 (18)	18 (20)	18 (20)
Rosenbrock	10 (14)	18 (20)	18 (20)	18 (18)
Rastrigin	20 (12)	20 (18)	20 (17)	20 (18)
Schwefel	14 (10)	11 (17)	18 (17)	11 (17)
Griewangk	20 (14)	18 (18)	20 (20)	18 (17)
Ackley	20 (16)	20 (19)	20 (20)	18 (18)

Table II shows the success rates for both non-shifted and shifted versions of the functions for the 8-population versions of each of the models tested in this paper. For example, on the Rastrigin function, model T1 achieved the target fitness on the non-shifted case in 11 runs out of 20, and achieved target fitness in the shifted case in 17 runs out of 20. All models perform well in comparison with a standard single-population EA (which otherwise uses the same operators and overall population size). This is not surprising, although it is important to confirm. The performance advantage over the serial model (the "1-client" column) is more pronounced when we consider the shifted versions of the functions. In terms of success rates, the relative performances of T1, T2 and T3 are very close, but perhaps T2 seems to have the advantage.

TABLE III. SUCCESS RATES (OUT OF 20) FOR 16-POPULATION VERSIONS (1-CLIENT INDICATES SINGLE POPULATION SERIAL METHOD WITH EQUIVALENT OVERALL POPULATION SIZE): NON-SHIFTED AND SHIFTED FUNCTIONS

Function	1-client	T1	T2	T3
Sphere	18 (20)	20 (20)	20 (20)	20 (20)
Rosenbrock	10 (14)	20 (20)	20 (20)	20 (20)
Rastrigin	20 (12)	20 (18)	20 (20)	20 (20)
Schwefel	14 (10)	16 (17)	17 (18)	15 (18)
Griewangk	20 (14)	20 (20)	20 (20)	20 (20)
Ackley	20 (16)	20 (20)	20 (20)	20 (20)

Table III shows us the results for the 16-population versions of these models. Generally the performance of T1, T2 and T3 are all a little improved, with T2 perhaps maintaining a slight advantage, but with no statistical significance based on these data alone. The "1-client" column is repeated here for convenience, but reflects the same benchmark comparison experiment reported in table II.

In table IV we can see the mean execution times of the runs that were successful in reaching target fitness. Now we can see a much clearer advantage for T2, which seems to outperform T1 and T3 in each case.

TABLE IV. MEAN EXECUTION TIMES (MS) OF SUCCESSFUL RUNS (AVERAGED OVER ALL SUCCESSFUL RUNS) FOR 8-POPULATION VERSIONS (1-CLIENT INDICATES SINGLE POPULATION SERIAL METHOD WITH EQUIVALENT OVERALL POPULATION SIZE): NON-SHIFTED AND SHIFTED FUNCTIONS

Function	1-client	T1	T2	T3
Sphere	72515 (5.14e6)	2261 (138503)	**1565 (112712)**	1802 (137969)
Rosenbrock	375703 (1.01e7)	246484 (1.76e6)	**193983 (1.40e6)**	313005 (1.80e6)
Rastrigin	35197 (1.44e6)	2162 (153054)	**1562 (114120)**	2172 (132423)
Schwefel	1.96e6 (1.59e7)	34059 (528545)	**25316 (426094)**	49369 (713984)
Griewangk	77949 (1.35e6)	2233 (165531)	**1542 (143793)**	1856 (163831)
Ackley	75139 (1.30e6)	2031 (141634)	**2000 (132812)**	2359 (146466)

In tables V and VI, we can see the speedup fractions for all cases. These simply divide the mean execution time of successful runs in the serial case by the mean execution time in the parallel model case. The tables provide these data for each function, and the best speedup for each function is given in bold. The final line provides an indicative "mean speedup" over all functions.

TABLE V. SPEEDUPS OF THE PARALLEL ARCHITECTURES COMPARED WITH SERIAL EA – 8-POPULATION MODELS

Function	Not Shifted			Shifted		
	T1	T2	T3	T1	T2	T3
Sphere	32	**46**	40	37	**46**	37
Rosenbrock	**2**	**2**	1	6	**7**	6
Rastrigin	**23**	**23**	16	9	**13**	11
Schwefel	58	**78**	40	19	**23**	14
Griewangk	35	**51**	42	8	**9**	8
Ackley	37	**38**	32	9	**10**	9
Mean	30.65	**46.76**	32.55	14.72	**17.99**	14.13

TABLE VI. SPEEDUPS OF THE PARALLEL ARCHITECTURES COMPARED WITH SERIAL EA – 16-POPULATION MODELS

Function	Not Shifted			Shifted		
	T1	T2	T3	T1	T2	T3
Sphere	87.55	**93.21**	89.78	100.40	**110.91**	103.19
Rosenbrock	9.38	**10.07**	7.28	13.48	**18.36**	14.09
Rastrigin	48.23	**51.20**	42.18	25.63	**32.72**	25.89
Schwefel	11.61	**22.93**	15.99	15.59	**16.40**	11.30
Griewangk	90.70	**111.04**	103.09	33.01	**45.30**	23.06
Ackley	92.86	**111.05**	93.55	28.64	**41.97**	25.03
Mean	57.23	**66.58**	58.64	36.13	**44.28**	33.76

It is clear that model T2 is the most successful in terms of speed of finding global optima in each case.

.

IV. CONCLUDING DISCUSSION

We have argued that distributed evolutionary algorithms (dEAs) are of ever-increasing interest and importance for a variety of reasons. It is well known that parallelized optimization can provide more advantages than simply speed of execution; meanwhile the design of a distributed, asynchronous architecture leads to opportunities for managing exploration and exploitation in ways that simply cannot be done in serial, and these can lead to better results for the same overall number of function evaluations. Given that parallel hardware resources are becoming more common and everyday, it is therefore clear that we need to understand how to design dEAs to optimal effect.

So far, however, there has been relatively little in terms of exploration of the vast number of potential architectures and migration strategies (for example) in the broad space of possible dEAs. In this paper we have compared some simple dEA topologies and interaction schemes. One was a straightforward case of dividing the population into *N* subpopulations, where a master process distributed new best chromosomes to the current worst subpopulation whenever a new best was found. The second was a slight variation on the first, in which the individual populations were each groups of two subpopulations, which communicated their best chromosomes regularly to each other, with the "best to worst" strategy operating at the level above these pairs of populations. The third model was based on Albe et al's cube topology [7] and also used the a similar migration scheme.

Interestingly, we found that the simple variation between model 1 and model 2 led to a significant difference in performance, which was clearly seen in the average speedups, in both the 8-population and 16-population cases. Model 2 appeared more successful than Model-3, inspired by Alba et al's GD-RGCA, on the functions tested.

One possible conclusion is that GD-RCGA (and perhaps other current models, might be enhanced by adopting aspects of the interaction strategy and topology used in model 2. This is one idea that we expect to examine in future work. Also in future work we will more systematically explore the design and parameter settings of model 2, to determine what seems to lead to its outperformance of model 1. We expect it will be interesting to explore this, for example, by tracking the generation and behaviour of niches in the fitness landscape as they emerge and are shared between processors.

REFERENCES

[1] T. Maruyama and T. Hirose and A. Konagaya, A Fine-Grained Parallel Genetic Algorithm for Distributed Parallel Systems. In proceedings of the 5th International Conference on Genetic Algorithms, pp. 184-190, 1993.

[2] S. Baluja, Structure and Performance of Fine-Grain Parallelism in Genetic Search. In proceedings of the 5th International Conference on Genetic Algorithms, pp. 155-162, 1993.

[3] H. M¨uhlenbein, Parallel genetic algorithms, population genetics and combinatorial optimization. In proceedings of the 3th International Conference on Genetic Algorithms, pp. 416-421, 1989.

[4] B. Manderick and P. Spiessens, Fine-grained parallel genetic algorithms. In proceedings of the 3th International Conference on Genetic Algorithms, pp. 428-433, 1989.

[5] T. Starkweather and L. Darrell Whitley and Keith E. Mathias, *Optimization Using Distributed Genetic Algorithms.* In proceedings of the 1st Workshop on Parallel Problem Solving from Nature, pp. 176-185, 1991.

[6] V. S. Gordon and L. D. Whitley, *Serial and Parallel Genetic Algorithms as Function Optimizers.* In proceedings of the 5th International Conference on Genetic Algorithms, pp. 177-183, 1993.

[7] E. Alba, F. Luna, A.J. Nebro, A.J., J.M. Troya *Parallel heterogeneous genetic algorithms for continuous optimization*. Parallel Computing. Vol 30, Nº 5-6, pp. 699-719. May/June 2004

[8] L. D. Whitley, "The GENITOR algorithm and selection pressure: Why rank-based allocation of reproductive trials is best," in *Proc. 3rd Int. Conf. on Genetic Algorithms.* San Mateo, CA: Morgan Kaufmann, 1989, pp. 116–121.

[9] Uwe Aickelin, Larry Bull: Partnering Strategies For Fitness Evaluation In A Pyramidal Evolutionary Algorithm. GECCO 2002: 263-270.

[10] Muhlenbein, H. and Schlierkamp-Voosen, D. (1993). Predictive models for the breeder genetic algorithm i. continuous parameter optimization. Evolutionary Computation, 1(1):25{49.