Parallel Genetic Algorithms For Optimizing Morphological Filters

4 authors, including:

Stephen Marshall
University of Strathclyde
201 PUBLICATIONS **5,394** CITATIONS
SEE PROFILE

John J Soraghan
University of Strathclyde
378 PUBLICATIONS **4,935** CITATIONS
SEE PROFILE

Neal R. Harvey
Los Alamos National Laboratory
78 PUBLICATIONS **1,229** CITATIONS
SEE PROFILE

PARALLEL GENETIC ALGORITHMS FOR OPTIMIZING MORPHOLOGICAL FILTERS

P Kraft*, S Marshall, J J Soraghan, N R Harvey

*Technische Universität Hamburg-Harburg
University of Strathclyde, Scotland

ABSTRACT

In this paper several design methods for parallel genetic algorithms (PGAs) optimizing morphological filters are discussed and an optimal scale and machine independent PGA for a loosely coupled, homogeneous or inhomogeneous multiprocessor computer is developed. The optimization is made in terms of computation speed, parallelization efficiency and quality. Quality means, in this context, the fitness of the best chromosome, which is an objective measure for the performance of the corresponding morphological filter in a particular environment.

INTRODUCTION AND PROBLEM BACKGROUND

Genetic algorithms (GAs) provide a class of robust stochastic search algorithms for various optimization problems. The application of GAs to the design of morphological filters for image processing has recently become an important area of research [Harvey(1), Ehrhardt(2), Chee-Hung (3)].

The standard GA model is based on the evolutionary process found in nature, applied to artificial optimization problems [Holland(4)]. A set of chromosomes (individuals) decodes all possible solutions for a given problem while an evaluation function assigns a fitness value to a given individual. Genetic operations such as crossover and mutation are used to modify individuals to achieve a higher maximum and average fitness in the given set of individuals (population). The basic GA can be described as follows:

```
create an initial random population
REPEAT generation
  select pairings for all individuals and perform
      genetic operations
  evaluate fitness values for all individuals
  select copies of best individuals for the
      next generation
UNTIL termination condition is true
```

Applied to the problem of filter optimization, the fitness value of an individual of the population (a chromosome representing a structuring element and a sequence of morphological operations) is calculated based on the performance of the associated filter, in a given task, such as noise suppression. Therefore, several filter operations have to be computed for each chromosome of the population, in every generation. The arithmetic complexity O for the calculation of a single fitness value results in:

$$O = (n \times n) \times (m \times m) \times M \ .$$

Here n refers to the dimension of the filtered images, m refers to the dimension of the filter mask and M is the number of performed morphological operations (erosion and dilation). This leads for average problems (images size 512x512 pixel, mask size 7x7 values and 4 morphological operations) to a calculation time of more than 30 second (on a T805 transputer or a Sun SPARCstation IPC), giving a total run-time for a normal sequential GA (one population with 100 individual and 200 generations) of about 7 hours. To achieve more acceptable computation times, which are required for real-time image analysis, it is necessary to use high performance parallel computers.

MODELS FOR PARALLEL GENETIC ALGORITHMS (PGAs)

It is expected that if a population of individuals is allowed to evolve in parallel then the underlying GA will be well suited for parallel execution. However, the standard GA cannot be simply parallelized in a straightforward fashion because of the requirement for global control in the selection step. It has been shown that a parallel implementation of the standard sequential GA, which simply distributes the evaluation process onto parallel working processing elements, cannot reach a satisfactory efficiency [Macfarlane (5)]. New structures of PGAs need to be designed to achieve this aim.

Parallelization methods

In general there are three methods of designing a PGA.

(*i*) **Subpopulations.** The most common method [Stender (6)] is to split up the population of individuals into subpopulations, for which the standard GA is used on different processors. These subpopulations exchange information about the most fit individuals using definite time intervals. Depending on the size of the subpopulation in relation to the entire population size, this method leads from a coarse grained distribution, with thousands of individuals within a single subpopulation, to a fine grained one, where each subpopulation consists of only one individual.

(*ii*) **Farming.** The second method is based on calculating the fitness values for a population in parallel. Using a farming concept, a master selects individuals from the population and distributes them to worker tasks on different processors. The workers perform the genetic operations, calculate the fitness values and return the result to the master, who continuously updates the population.

(*iii*) **Data parallel.** The third method is based on splitting the calculation of a single fitness value. A set of processors calculates the fitness of an individual for different parts of the image. In a second step these values will be combined to achieve the real fitness value.

In all three methods no image data has to be transmitted after the initialization process has finished, except if the problem is changing, i.e. adaptation to a changing environment, and only small packages of data related to individuals (chromosomes and fitness values) are transferred between processors during the main process. The computation to communication ratio is very large. This implies that GAs will have a good parallelization efficiency even on processor networks with slower connections such as workstation clusters.

Hardware requirement and scale-ability

The available parallel computer is the deciding factor for the choice of the appropriate parallelization model. Usually a combination of more than one is used. Splitting up the entire population into subpopulations leads to better optimization results [Stender (6)] but does not reduce the run time by an efficient factor, because each subpopulation still has to reproduce itself a couple of times until a desired improvement in the fitness values results. If the number of subpopulation rises over a certain level it will be more likely to have similar subpopulations searching for the same minimum on different processors, leading to redundancy and poor efficiency. The communication network demanded for the subpopulation method needs to deal with a load depending on the time it takes to calculate a single fitness value and the number of subpopulations. Experimentally it was found that each subpopulation should be able to send a message of $10^2 - 10^3$ bytes of individual and additional information at least after a number of calculated fitness values which is equal to its population size. If the processors are connected with a single bus, the available bus capacity should equal the sum of the requirements of all subpopulations.

The farming method is useful for further parallelization in addition to the subpopulation method. It can be used with a reasonable efficiency up to a number of processors less than half of the population size where only individuals for the current generation can be calculated in parallel. This causes the maximal achievable speed-up factor to been less than the number of individuals within a population. It was experimentally found that the connection between the master and each worker should be able to transfer a message in a time slot which is less than 10 per cent of the time needed to calculate a single fitness value.

If a single processor cannot cope with the problem of calculating fitness values, for example, if the available memory is not big enough to store a whole image or further processors are available, than the data parallel method can be used. Then the maximum number of efficiently used processors depends strongly on the size of the image and the communication capacity between the processors. It is also possible to use a pipeline concept for calculating the sequence of filter operations on different processors. This has the disadvantage of the filtered images being transferred between different processors, which is only effective if the pipeline elements are connected with a high speed communication structure.

In general it has been found that the total amount of loosely coupled processors which can theoretically be used to calculate an average optimization problem with an acceptable efficiency sums up to at least 10^5 processors. This number is based on experiments which have shown that $10^2 - 10^3$ is a satisfactory value for the numbers of subpopulations with a population size of approximately 10^2 individuals. The appropriate number of processors calculating a single fitness value with an efficiency above 0.85 results in 10^1 using a transputer network.

Communication and Synchronization

Earlier approaches in PGAs have used fixed time points for the communication within a population [Stender(6)], such as a communication step at the beginning of each generation. For example:

```
create an initial random population
REPEAT generation
  exchange individuals with other subpopulation
  select pairings for all individuals and perform
      genetic operations
  evaluate fitness values for all individuals
  select copies of best individuals for the
      next generation
UNTIL termination condition is true
```

This method has to use blocking message passing functions which could lead to idle processors if the execution time for a generation differs between subpopulations. This could be caused by (i) a different processor load, a different processor speed or a different number of processors working on a subpopulation or (ii) a different number of instructions required to calculate subpopulations. The latter can be caused by different population sizes or different times needed to calculate a single fitness value. This is due to the fact that individuals can differ in their number of morphological operations and that for offspring identical with one of their parents the fitness value is simply copied and not calculated a second time. Another disadvantage of this communication scheme is that the feature of concurrent communication and processing which many processors and message passing systems offer, cannot be used. A new communication scheme has to be developed which does not suffer from the above outlined disadvantages.

Virtual parallel communication A way to solve these problems is by using concurrent communications. All inter-processor communications are implemented by tasks, running in parallel to the execution of the processing of a generation (performing genetic operations and calculating fitness values) on the same processor. These tasks can queue outgoing messages until a communication channel is ready to communicate or buffer incoming messages during the time a generation is processed until they can be integrated into the receiving subpopulation. This means that individuals from different generations (with more or less improved chromosomes) can be put together in the same generation which is even more appropriate to the natural concept of evolution. Depending on the organization of the population it might be possible to integrate new individuals at any time, without waiting for a processed generation. In this case further management overhead could be saved.

GENERAL PARALLEL GENETIC ALGORITHM (GPGA)

The developed GPGA uses all the advantages of the above described methods and differs in some points (fixed generations, fixed population size, homogeneous communication scheme) form GAs and PGAs proposed so far. Even for the standard algorithm the theoretical background is not fully understood and only a few references exists to describe their behaviour [Whitley (8)]. The GPGA is more similar to natural evolution, which seems to be a more appropriate solution.

Population organization.

A population is represented as a list of individuals ordered in terms of fitness values. Each individual has a rank defined by its position in the list: the lowest for the last individual (worst fitness value) and the highest for the first one (fittest individual). At any time new individuals can be added to the list - sorted into the right position - or deleted, which may change the rank of other individuals. The access to the list of individuals is organized with a semaphore concept so that different tasks can concurrently add, copy or delete individuals. Due to this the entire number of members within a population can change at run-time. A control task monitors the population size and deletes individuals from the population if the population size grows over a threshold. The selection of individuals to be deleted is made at random with a higher probability for individuals with a lower rank to be selected, which is a survival of the fittest strategy.

Selection and reproduction.

The probability for individuals being selected for reproduction is made referring to their rank and not to their fitness value, by selecting parents at random while using a skewed probability distribution preferring fitter individuals to be selected. This has the advantage that no fitness scaling is necessary to overcome the problem of unpleasant fitness distributions, like an accumulation of good fitness values close together and some poor values at the far end of the fitness range. Individuals selected as parents are copied and genetic operations are performed to produce offspring. The fitness value of the offspring is calculated and they are sorted back into the population which has now increased by two individuals. This procedure of reproducing is repeated until a termination condition stops the algorithm. This mechanism allows individuals from different generations to crossover or the same parents to have more than one pair of offspring.

Parallelization levels

The first level of parallelization is based on a number of subpopulations which are calculated from four different types of tasks in parallel. The first type of task selects a couple of individuals as parents and performs the genetic operations on them, calculates the fitness value for the offspring and sorts them back into the population. Several of these tasks working in parallel on different processors constitute the second level of parallelization. The second task is collecting messages from other subpopulations. These messages contain a single individual, its fitness value and time value. The individuals are sorted into the ordered population and the time value is used to modify the optimal population size. The size for each population is allowed to vary in a given range. If the time value, which represents the average time needed to calculate a fitness value times the actual population size for the processor sending the message, is less than the value on local processor, the population size is decreased otherwise it is increased. This mechanism implies a load balance to achieve a soft synchronization between subpopulations. The third task is transmitting copies of individuals to other subpopulations. The selection from the population is made in the same way as the one for selecting parents. The target population is chosen at random from a given set of subpopulations. Depending on the network this set could contain only local neighbourhood populations (for a grid array) or all subpopulations (for bus networks). The ratio of sending individuals is controlled with a function having the calculation time for a single fitness value and a free eligible value as parameter, so it can be adapted to the available communication power of the network used. The fourth task is the one of supervising the population size. If an appropriate parallel computer is available in the third level of parallelization each of the tasks calculating the fitness values (type one) can employ more processors for calculating the fitness values (third parallelization method) of the offspring. The developed algorithm uses two processors to calculate the fitness values for both offspring of a couple in parallel.

EXPERIMENTS AND RESULTS

The numerical complexity and the behaviour of the developed PGA has been investigated and the results have been compared with results of a standard parallel genetic algorithm (SPGA)and a coarse grained parallel genetic algorithm (CPGA). The standard algorithm is basically a sequential one, which evaluates the fitness function for all chromosomes in parallel. A master task distributes couples of individuals to worker tasks, which perform crossover, mutation and calculate the fitness values. The master task collects all individuals, determines the number of copies for each individual and chooses new couples to be sent out. The coarse grained versions consist of distributed populations with particular communication schemes. The individual populations are processed concurrently on different processing elements and after a number of generations the separate populations export a set of populations to other neighbouring populations.

Parallel Hardware. The algorithm is implemented on three different types of parallel computers. The first is an inhomogeneous network of transputers, whose connection structure is varied in several tests. The single processor elements differ in the type of the transputer used (T400, T800, T805), their operating frequency (17.5 MHz, 20MHz, 30 MHz), their memory access time (0 to 4 wait states) and their semiconductor memory capacity (256 kbyte - 4 Mbytes). The second computer consists of a homogeneous grid of 128 identical T805 transputers modules (30 MHz, 4 Mbytes). Finally, a cluster of more than 50 Sun SPARC workstations, shared with other users, is used as one parallel machine. The overall amount of available computation power for the PGAs on this cluster changes between different test runs and also during one run.

Software Environment. The algorithms are written in ISO C and make use of special libraries for parallel processing. While on the workstation cluster only PVM was used on the homogeneous transputer network and parallel image processing library (PIPS) [Bienieck (7)] was supplied next to other environments (Par C, 3L C).

Search efficiency. In all tests the GPGA produced a better behaviour than the SPGA. This means it achieves a better quality of filters after the same amount of overall calculated fitness values. Compared with the coarse grained algorithm the results have the same quality or are even better using the same distribution of subpopulation. Fig. 1 shows the maximal achieved fitness values for the three algorithms for different number of calculated fitness values. The number of subpopulations used for the coarse grained algorithm and the GPGA is raised with the amount of calculations. The hardware environment for all tests is the homogeneous transputer system.

Parallelism efficiency. To compare the efficiency all algorithms were run on the same transputer network, using 4, 8, 16, 32 and 64 processors. The number of fitness calculations was kept fixed for all runs. The timings in seconds are given in Fig. 2. The processor efficiency graph corresponding to the timings is shown in Fig. 3. One of the advantages of the GPGA is the ability of using inhomogeneous processor networks while the other cannot cope with them in a satisfying way. Fig. 4 shows the average of calculated fitness values per minute for the GPGA running on different numbers of workstations. The efficiency is not easy to calculate because of the differing speeds of the processors used and their loads varying in time.

CONCLUSIONS

Several methods to parallelize GAs have been discussed, implemented and compared. A new model of organizing populations, allowing them to grow by being supervised from a task deleting worst individuals has been introduced. A GPGA avoiding idle processors has been developed and tested on various parallel computer systems. Considering the superior results on all types of networks, even on inhomogeneous networks of workstation clusters, it is concluded that the GPGA is a useful tool for designing methods of morphological filters, and also for users not having access to massive parallel computers.

An important part of future work will be the use of different processor types such as PowerPC-processors or TMS320C40 digital signal processor in networks with more than 10^2 processors.

REFERENCES

1. Harvey N, Marshall S, 1994, "Using Genetic Algorithms in the Design of Morphological Filters", Mathematical Morphology and Its Application to Image Processing, 53-59

2. Ehrhardt R, 1994, "Morphological Filter Design with Genetic Algorithms", Image Algebra and Morphological Image Processing, SPIE Vol. 2300, 2-12

3. Chee-Hung H C, 1989, "A Genetic Algorithm Approach to the Configuration of Stack Filters", ICCA89, 219-224

4. Holland J H, 1992, "Adaptation in Natural and Artificial Systems", MIT Press, Cambridge

5. Macfarlane D and East I, 1990, "An Investigation of several Parallel Genetic Algorithms", Proc. of the 12th Occam User Group, 60-67

6. Stender, J. 1993, "Parallel Genetic Algorithms: Theory & Applications", IOS Press, Amsterdam, Netherlands

7. Bienieck A, Jungclaus N, Nölle M, Schreiber G and Schulz-Mirbach H, 1994, PIPS 2.1, Parallel Image Processing System, Technische Universität Hamburg-Harburg

8. Whitley D, 1993, "A Genetic Algorithm Tutorial", Department of Computer Science Colorado State University

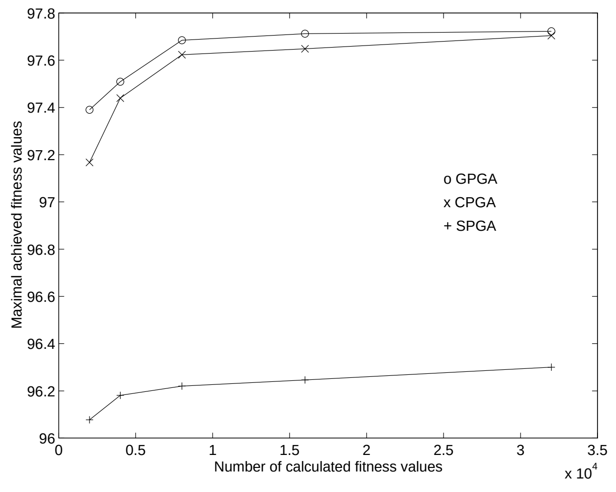

Figure 1: Search efficiency

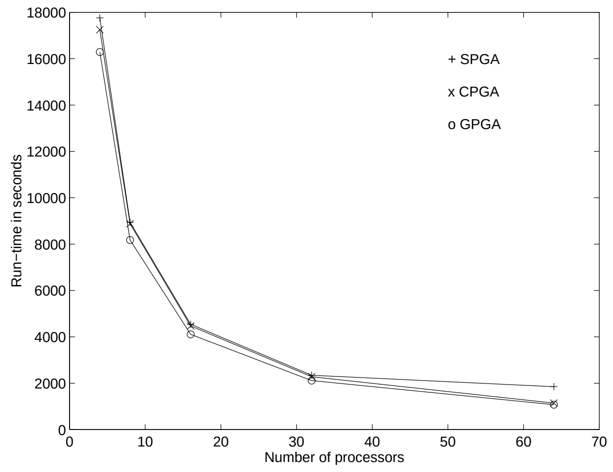

Figure 2: Run-time on a transputer network

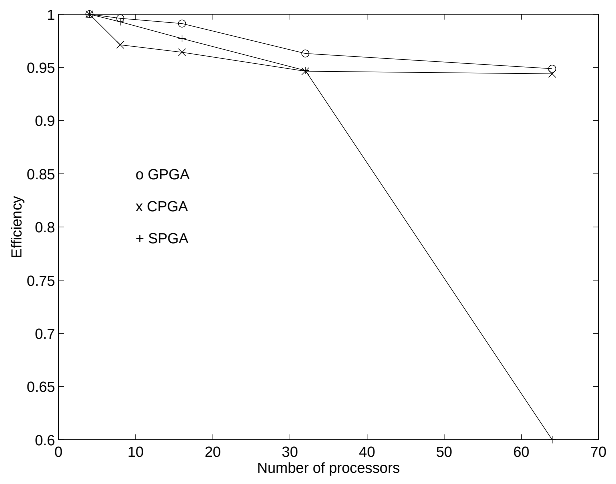

Figure 3: Parallelization efficiency

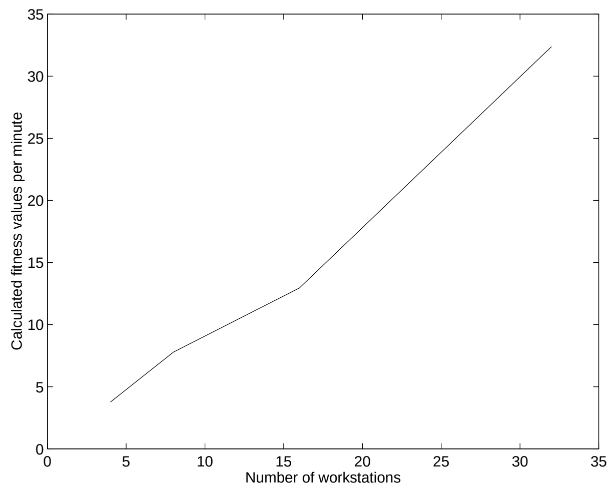

Figure 4: Speed on a workstation cluster