Parallel genetic algorithms in a networked workstation environment

Vítor Oliveira, António Pina, Alberto Proença
Departamento de Informática, Universidade do Minho, Campus de Gualtar, 4710 Braga, Portugal
Email: vspo@ci.uminho.pt, {pina,aproenca}@di.uminho.pt

Abstract

Parallel Genetic Algorithms are suited to deal with problems with very large solution spaces and they can support efficient parallel distribution of work. In a PGA Island Model the migration strategy can take advantage of high latency communication channels in a distributed system. This approach suggests the use of networked workstation environments as a cost effective alternative to MPP systems. A Genetic Algorithm Programming System (GAPS) was developed to evaluate the proposed approach, which supports the design of parallel genetic programs and its execution in a distributed workstation environment. GAPS separates the specification of the problem and the user application interface, from the implementation and management details of the run-time environment; it also addresses fault tolerance, needed to recover from a fault that may occur in a dynamic network of heterogeneous workstations. GAPS uses PVM to implement a structural load balance strategy, which distributes complex evaluation functions with large chromosomes across a parallel machine. The proposed system showed to be effective when tested with the knapsack problem.

1 Introduction

The parallel nature of Genetic Algorithms (GA) [1] is a well known feature that the research community is aware of; the benefits include faster and more robust solutions with better quality. Current research on Parallel Genetic Algorithms (PGA) [3, 4] emphasises its development to take advantage of Massively Parallel Programming (MPP) systems, based on multiple processing nodes interconnected by very fast communication links.

The Island Model is one of the most popular models among PGAs[2, 5, 6]. In this model an initial chromosome population is distributed across the available computing nodes, the islands. At each island, a sequential genetic
algorithm implements the evolution of the local population; the global
distribution of the intermediate local solutions is performed across the islands
through a chromosome migration.

Evaluation studies of the Island Model show [2, 7] that the convergence
speed and the quality of the obtained solutions do not necessarily rely on a high
migration rate per island. This fact suggests that the use of distributed systems
with higher communication latency and lower bandwidth than MPP systems
may provide a feasible and low cost alternative.

The generalisation of distributed computing environments, such as PVM
(Parallel Virtual Machine) [18], also contributed to explore that alternative.
Most research teams can now afford to experiment on virtual parallel systems
with high computing power and at low cost. The work being presented here
aims to show the feasibility to use a network of heterogeneous workstations as
an effective alternative to study and develop PGAs.

To fulfil this goal a tool was developed - a Genetic Algorithms
Programming System, GAPS - which assists the design and the run-time support
of genetic algorithm programs in a parallel and distributed environment. The
programs that are developed using this tool show some relevant features, such as
an easy programming interface, fault tolerance and an adequate management of
the available computing resources. The programming environment supplied with
GAPS lets the user concentrates on the problem details itself and on the search
for the appropriate evaluation functions, and at the same time frees the user from
the computing management tasks under dynamic conditions, on the parallel
virtual machine.

This communication opens by presenting the most common PGA models,
and by assessing some of the problems related to their parallelisation in a
networked workstation environment. The section that follows aims to describe
the proposed GAPS, from a generic introduction to its programming support and
run-time environment. It then follows by an preliminary evaluation from
gathered data, and closes with suggestions for future work.

2 Parallel Genetic Algorithms

The computational model based on Genetic Algorithms (GA) [1, 8] is
inspired from the natural evolution processes. The application of GAs to
optimise combinatorial problems have shown its capability to solve high
complex problems in irregular and multimodal solution spaces, by combining
the space exploration of solutions with an adequate selection of the best results.

The analogy between GAs and the natural evolution is expressed at the
problem specification through a chromosome population and a set of genetic
operators: selection, mutation, elitism, local hill-climbing, ... When an adequate
computational encoding is found for a specific problem, the genetic operators
and the evaluation function are applied; then the next step is to find the set of
control parameters that will lead the GA to converge to an optimum solution at
the shortest time.
The great advantage of GAs when compared to other optimisation techniques [9] lies on the high speedup values that can be obtained from parallel GAs, that may eventually result on a linear speedup [7]. In some circumstances - niching and speciation theory - this advantage is even greater since the quality of the obtained solutions from the parallel algorithm can be higher than the one from the equivalent sequential algorithm [2].

2.1 Models

The most common parallelisation techniques are also inspired from nature. The diagrams in figure 1 show how the island model and the neighbourhood model can be used to specify a problem.

![Diagram of PGA models: a) the island model b) the neighbourhood model](image)

The neighbourhood model [11] follows the analogy of long territorial extensions: the entities interbreeding is achieved almost exclusively among the local neighbours. The computing approach allocates a chromosome to each processing node, and the interbreeding is between neighbour processors. This approach leads to a slow spreading of the solution across all interconnected processing nodes.

The global population in the island model is distributed among the islands [11, 13]; each entity can interbreed with any other in the island. The definition of a chromosome migration policy between islands will allow a proper dissemination of the chromosome characteristics - obtained by the local evolution - across the islands.

Although the quality of the solutions that are obtained from these two models are equivalent [7], their computational results may be different. As an example, since the neighbourhood model is strongly dependent on the interprocessor communications to produce mature solutions, it is not appropriate to implement this model on a parallel or distributed environment with long latency communication channels.

When the island model is used, some precautions are required to avoid a premature convergence of the genetic algorithm - due for instance to a high migration rate - which may result on solutions that did not fully explore the search space, and as a consequence the solutions may show a lower quality grade. Similar situations may occur due to a wide interconnection topology -
even if the partial solutions are able to propagate faster - since it reduces the population diversity [15].

2.2 Implementation on a workstation network

Empirical evaluation studies have shown [7] that the migration rate can not be too high to affect the diversity richness, nor too low to not support solution sharing. Experimental results suggest that the migration rate could be lowered provided the amount of chromosomes was increased per migration, and vice-versa.

The island model can be adapted to a parallel environment with low requirements on communication speeds, provided the following factors are considered when defining the migration policy: (i) neighbourhood topology, (ii) migration rate, and (iii) number of entities per migration.

To keep the quality of the obtained solutions, some migration policies could be followed which minimise the migration flow: to wait for the stabilisation of a local population before pursue the chromosome migration [21] or to lengthen the time intervals between migrations.

The computational solution with the island model may also benefit from the implicit concurrency between the two independent processes [16]: the computational tasks - the execution of the sequential genetic algorithms - and the communication tasks - the chromosome migration.

A typical MPP system is based on a pre-defined number of computing nodes, that are interconnected according to a static topology (some current MPP systems are already based on dynamic interconnection topologies). All nodes have similar computational features, with equivalent performance ratios, and the interconnect topology has fixed and predictable communication delays.

On the other hand, a network of heterogeneous workstations may present a completely different behaviour: each computing node may vary from a uni-processor system to a SMP one, may be based on different processors which may run at different clock speeds and with different memory sizes, each managed by a different operating system, and even the network configuration itself may vary dynamically and be prone to logical or technical faults. Similar considerations also apply to the underlying communication system - both at the physical and the logical protocol levels - and also in relation to an eventual fault situation.

These considerations suggest that, to obtain reliable results on a network of heterogeneous workstations, the implementation strategy should follow some strict rules, including support to fault tolerance and an adequate load balance mechanism. The compliance to these rules can enforce the use of a network alternative - versus MPP systems - that already benefit of its openness, low cost and increasingly higher computational power.

3 A Genetic Algorithm Programming System (GAPS)

The Genetic Algorithm Programming System (GAPS) being developed aims to fill a gap between the design of PGAs and its execution in a general
purpose distributed system. These goals are present at the GAPS architecture definition (figure 2), which makes a clear distinction between the design of a solution - the problem specification and the interface definition - and its execution - including the management of the available concurrent resources.

3.1 GAPS overview

Any programmer should concentrate its effort on the design of a solution, and to use genetic algorithms he/she should also code its specific characteristics. To build a program those functions are linked to another set of modules - supplied by GAPS - which allow the resulting program to automatic and dynamically take advantage of the available run-time execution environment.

The resulting program follows a structural load balancing strategy, by using PVM primitives to create and distribute islands across the computing nodes at the virtual machine. This strategy supports fault tolerance through mechanisms that manage the faults due either to a technical fault at a workstation or a communication line, or to a logical flaw caused by a dynamic modification of the parallel system configuration.

Applications

The applications layer contains the programs that interface the user to the Genetic Algorithm Engine (GAE). These programs interact with GAE to define the global behaviour of the system and the working conditions, and to monitor the results.

The complexity of the applications is only limited by the set of operations that are defined by the interface commands.

The system supports concurrent execution of multiple applications; each of these may aim different goals, and no distinction is made between interactive an non-interactive applications.
Interface

The commands at the interface maintain the link between the applications layer and the GAE. The defined operations at this level were selected to act as a communication link between the applications and the GAE.

When the commands are activated, the applications may interact with the program to perform enquiry or modify operations. The set of current basic commands in GAPS may be extended at any time, to provide further facilities.

Genetic Algorithm Engine (GAE)

The parallel machine is controlled by the GAE, which also controls the execution of the parallel algorithm. The GAE acts as an archipelago composed by all the islands that belong to the same genetic algorithm.

All islands follow the same working pattern and they all contribute to the solutions, within reasonable autonomy limits. A global control strategy over the archipelago is established by the GAE, which also manages the parallel execution of the algorithm in a transparent way to the user. This centralised action of the GAE is performed on the island that, at the time of its execution, holds the flag of main controller (figure 3). The GAE also manages the computing resources at the parallel virtual machine, and monitors the model global state and each island local state.

The computing island model

The islands computing structure in GAE can be represented by the functional diagram in figure 3. Beyond the control functions, all activities at each island are oriented towards the actions on the population. The genetic algorithm being used is based on GENITOR [17], but with a different migration policy: GENITOR propagates copies of the chromosomes, while in GAPS the original chromosomes are the ones that travel.

The population evolution is accomplished through combined computing and migration processes. The computing process applies the sequential genetic
operators, one after the other, while the migration process takes care of the entities interchange policy among the neighbour islands. Another process controls the island functions, namely the required functions related to the inner working environment, the interface to the application layer, and the co-operation between islands to guarantee a global behaviour of the PGA.

All these processes can execute concurrently, provided the simultaneous access to the data structures representing the population are adequately controlled. This situation can occur due to the steady-state model being adopted to manage the population evolution, which uses incremental steps. The alternative generation model would block all accesses to the population while a new generation was being created.

3.2 The application interface

GAPS programs are formed by the genetic algorithm, the interface module and the GAE. To create a new program, all a programmer needs to do is to design and to code the corresponding evaluation function, related to a given problem.

```c
float eval(void * chrom, ...){ //Code and evaluation function
int main(){
... 
    agp_config( eval, ...); // start-up 
    if ( agp_init() == 0){ // switch on GAE
        { 
            agp_start(); // algorithm begins
            ... wait
            chrom = agp_getchrom( 0, & val ); // get the best chromosome
            printf( "%s = %f
", cvtxt(chrom), val );
        }
        agp_shutdown(); // switch off GAE
    }
}
```

Figure 4: Program code showing the interface module

Through the interface module the user can then define the goals to aim when executing the program, including the specification of input parameters and the output data. The modular structure of the interface block allows two types of interaction with GAE: at the program level, and through external applications that share the functions library in GAPS. The code in figure 4 exemplifies this.

Included in GAPS, the gaps command is an example of an external interface module, that can asynchronously interact with a running genetic program, as a console. Through this interactive application (see figure 3) - that can be launched from any workstation at the virtual machine - the user can monitor the program execution state and modify the program evolution, either at the genetic algorithm level, or at the computing resources.

3.3 The run-time support

GAPS was designed specially to take advantage of a workstation network. It allocates by default a computing island to each workstation, although more
islands can be allocated per workstation. This strategy guarantees the quality of the solutions, since it caters for both situations: when the total number of available workstations is short, and when each workstation has more processors and can better explore parallelism.

GAE also takes advantage of the PVM ability to complement the distributed operating system functions, to get the required computing resources to launch new islands or to dynamically adapt to the modifications of the network virtual machine. PVM provides communications support between the GAPS layers, both to interchange commands between applications, and to make direct contact with the islands. The data moving from island to island contains not only migrating chromosomes but also the required commands to implement the archipelago management policies.

Fault tolerance

When running long genetic programs - those that take a long time to converge - the number of available workstation may vary during that time. This may occur either due to the switching on/off of a workstation (logic fault), or due to any failure at the computing system or in the communication infrastructure (technical fault). To deal with these occurrences - without disrupting the running program, or without affecting the quality of the results - a fault tolerance and recovery strategy is required. GAPS uses a group communication strategy [20] to guarantee a coherent program execution, as long as an island is alive. When the main controller island fails, a new island takes control over GAE.

4 Experimental results

Measurements were taken to show the efficiency of PGAs in a network workstation environment. The results presented here aim to show that superlinear speedups can also be obtained in a distributed environment, and that both types of parallelism - inherent to the genetic algorithm itself, and the effective load distribution across the network - contribute to the super-linear speedup.

A NP-complete combinatorial problem was chosen to evaluate GAPS: to fill a weight-bounded knapsack with 64 packets, each with a different value and weight, such that the evaluation function - the maximum total value in the sack - is maximised (optimal known value: 98.5099).

All run tests used a fixed population size - 180 chromosomes - and a continuous migration flow with a fixed migration size - 6 chromosomes per trip. The measurements were taken at 5 seconds interval from a randomly chosen island. The values in the graphs are the arithmetic average of sets of 8 run tests; the trend lines follow a logarithmic function with a least square fit.

Graph a) in figure 5 show how the solutions quality vary with the number of workstations, allocating 1 island per node; the curves represent the measured fitness evolution and their respective logarithmic fit curve for each one of the four degrees of computing parallelism.
Graph b) in figure 5 distinguishes both forms of parallelism that contribute to the overall speedup; a second set of measurements were taken that preserved across the same four computing environments the overall amount of islands (12 islands). The dots in the graph resume the time measurements to reach within 5% error of the optimal, and their relative position to the linear speedup curve.

![Graph](image)

**Figure 5** Experimental results a) the quality evolution; b) speedup with fixed fitness value

It can be seen from figure 5 that when the number of islands is kept fixed - i.e., the algorithm parallelism does not increase with the number of computing nodes - the obtained speedup is slightly below a linear function. It can be concluded that the parallelism due to the genetic algorithm is crucial to obtain super-linear speedup.

5 Conclusions

A parallelisation policy applied to genetic algorithms based on the island model can take advantage of parallel and distributed computing systems. The programs that are created with the system being developed, GAPS, contain the desired parallel programming features of concurrency, modularity and locality.

This work showed that the use of a workstation network as a platform to study and develop PGAs based on the island model can be a viable alternative to MPP systems, in terms of cost, availability, execution time and efficiency.

The use of a heterogeneous and unreliable network environment can, however, impair the efficiency of PGAs. GAPS is more than just a user-friendly programming development system, since it also addresses the fault tolerance issues above mentioned, and manages the available computing and communication resources. The use of the PVM distributed environment together with the group communication strategy helped to add scalability to those features; this is particularly relevant when the overall computing capacity is heavily dependent on the available workstations on the network at each moment.

More experimental results are still required to evaluate and compare the quality and speed of the solutions provided by GAPS in a MPP environment and current implementation with a reasonable amount of heterogeneous workstations.
References


[20] Birman, K., The process Group approach to reliable distributed computing, Department of Computer Science, Cornell University, 1993

[21] Foster, I., Designing and Building Parallel Programs, Addison-Wesley Publishing Co. 1995