#7 — NOV. 95

EPFL

SUPERCOMPUTING REVIEW

NUMERICAL SIMULATION FOR SCIENCE AND TECHNOLOGY

Parallel Genetic Programming Induction of Binary Decision Diagrams

by Christian Miccio and Eduardo Sanchez, Laboratoire de Systèmes Logiques - EPFL & Marco Tomassini, Centro Svizzero di Calcolo Scientifico, CH-Manno and Laboratoire de Systèmes Logiques - EPFL

Genetic programming is a new technique for machine learning, program induction and optimization loosely based on an evolutionary paradigm. Genetic programming is easily amenable to parallel computing which help relieve the intrinsic slowness of the approach. We describe a parallel implementation of genetic programming on the T3D computer. We apply the system to a problem of induction of binary decision diagrams used in logical circuit design. It is shown that the results depend in a critical way on the representation of the decision diagrams and that the parallel implementation is able to find the correct solution with less computational effort than the sequential version.

Summary

Parallel Evolutionary Computing

Evolutionary Algorithms (EAs) are a class of artificial adaptive processes that find their origin and inspiration in the biological natural selection mechanisms. Genetic algorithms (GAs) [11] seek optimal or near-optimal solutions to hard search and learning problems by giving more chances of survival to fitter individuals in an evolving population in which each individual represents a feasible solution to the given problem through a suitably coded string of symbols. New solutions are explored by mixing good individuals and artificial mutations are used to prevent premature convergence to local optima by randomly sampling new points in the search space. Evolutionary algorithms have found increasing application to many problems in diverse areas such as hard function and combinatorial optimization, neural nets evolution, routing, planning and scheduling, management and economics, machine learning and robotics and pattern recognition.

Genetic programming (GP) is a variation of genetic algorithms in which the evolving individuals are themselves computer programs instead of fixed length strings from a limited alphabet of symbols [6]. Programs are represented as trees with ordered branches in which the internal nodes are functions and the leaves are the so-called terminals of the problem. The search space in genetic programming is the space of all computer programs composed of functions and terminals appropriate to the problem domain.

Suitable functions and terminals are determined for the problem at hand and an initial random population of trees is constructed. The population then evolves with fitness being associated to the actual execution of the program and with genetic operators adapted to the tree representation. The crossover operation first selects a random crossover point in each parent tree and then exchanges the sub-trees, giving rise to two offspring trees. Examples related to our application will be given in the next section. There are also provisions for preventing trees from becoming too deep, for simplifying trees and for compressing trees that perform a useful functions into a single reusable module.

Genetic programming is well-suited to parallel implementation. The most popular parallel models are the fine-grained or *grid* models, and the coarse-grain or *island* models. In the grid models, large populations of individuals are spatially distributed on a low-dimensional grid and individuals interact locally within a small neighborhood. In the island model the population is subdivided into smaller subpopulations which evolve independently and simultaneously according to a standard EA. Periodic migrations of some selected individuals between islands allow to inject new diversity into converging subpopulations. Micro-processor-based distributed memory machines and workstation clusters are well adapted for the implementation of this model. The advantage of parallel EAs for difficult problems is that they can handle larger populations in reasonable times and favor cooperativity in the search for good solutions ([5], [15]). In the following section we introduce the binary decision diagrams optimization problem and explain in more detail the parallel GP implementation used to solve it.

go to the summary

Genetic Programming for Binary Decision Diagrams Optimization

Binary Decision Diagrams

A binary decision diagram (BDD) is a type of oriented graph used notably for the description of algorithms. It assemblies, according to some rules, two types of nodes: the decision or test node and the output node. The decision node is equivalent to an if-then-else instruction: it realizes a test on a binary variable and, according to this value, indicates the node following. The output node produces a value. The two rules of assemblage are: there is one and only one initial node (the entry point of the algorithm); the output point of a node can be connected to only one entry point of an other node.

A binary decision tree is a binary decision diagram that respects a third rule of assemblage: any entry point of a node is connected to only one preceding node.

Since [8] it has been demonstrated that all logical boolean function can be represented by a binary decision diagram. This type of representation finds applications in the test and the implementation of logical functions [3]: the function of a decision node can be implemented by a multiplexor or demultiplexor circuit and a binary decision diagram can be implemented by an interconnection of these circuits. In all these cases, the minimalisation of the number of nodes used is important, for the cost and/or the time of execution of the function. Nevertheless, the complexity of this minimalisation is such that in most cases approximate solutions are accepted [12].

A renewal of interest on the minimalisation of binary decision diagrams is born with the appearance on the market of programmable circuits named FPGA (Field-Programmable Gate Array) [13] [10]. These circuits appear under the form of an array of identical cells (the logic cells), where the user can program the function inside every logic cell (among some possible) and interconnections between cells. Each FPGA manufacturer proposes a different type of logical cells and interconnections. Some, as Actel [1], propose very simple cells, formed of a simple multiplexor circuit. A minimal binary decision diagram can therefore drive to an optimal utilization of the FPGA cells.

go to the summary

The Genetic Programming Representation

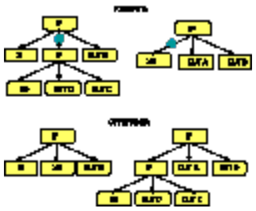

Figure 1

In preparing to use genetic programming to solve a problem one has to decide on the set of terminals, the set of primitive functions, the fitness measure, the stopping criterion and the values of some parameters such as population size and crossover rate [6]. The fitness of an individual is defined in our case to be the difference between a perfect solution and the actual number of hits of the given individual on all the input combinations of values. Therefore, a fitness value of 0 means that the individual correctly solves the problem.

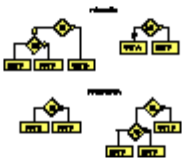

Figure 2

While the choice is seldom unique, the nature of the problem suggests suitable terminals and functions. For the representation of binary decision diagrams we tried two representations. In the first one the terminal set was made by the operation codes and the output codes. The only function operating on those terminals was the three-branches IF function. Random-constructed trees are not guaranteed to be valid decision diagrams for our problem since the IF function itself can be the first argument (i.e. the test condition) of another IF function. Furthermore, there is nothing to prevent an output code from also being the first argument to the IF function. Finally, input variables can only appear as test conditions. Likewise, crossing-over valid trees will not necessarily yield admissible offspring (see fig. 1). We therefore penalized invalid trees by giving to them the worst fitness value in order for them to be less likely to be selected for reproduction.

In the second solution we avoided mixing terminals by creating a specialized IF function for each operation code. Example diagrams and a crossover operation are shown for a simple case in fig. 2. With the latter choice of functions and terminals all trees are guaranteed to be valid. Results of the runs of parallel genetic programming using both representations will be discussed in section 3. We now briefly describe our parallel GP implementation choices.

go to the summary

The Parallel Implementation

The parallel architectures that best matches the rather coarse grain and variable length of genetic programs are micro-processor-based distributed memory machines, including workstation clusters. On these machines it is easy to implement the island model. Good results were obtained in [9] and [7] using a similar computing architecture.

The T3D multicomputer [4] is a DEC/Alpha-based MIMD machine. The processors are connected by a fast bidirectional 3-D torus interconnect network.The memory of the machine is physically distributed although, depending on the programming model used, it can be globally addressable. Communication latency is low and bandwidth is high due to latency hiding and data transfer optimized hardware and easy routing mechanisms. The T3D array is connected through I/O nodes to a Cray Y-MP host machine on which all program development takes place. Access to peripherals such as disks, tapes and the network is through the host. Three different programming models are available on the T3D: message-passing, work and data sharing using a global address space (CRAFT) and data parallelism.

The message-passing approach perfectly suits the island model for parallel genetic programming. It is based on PVM which is a standard message-passing environment.

We started from the publicily available sgpc GP program [14]. The PVM-based code parallelization was easy except perhaps for the modifications needed to pack and unpack program trees to be sent to other subpopulations (islands). This requires linearizing the trees to be packed in a message buffer and rebuilding them at the destination subpopulation in the new processor's private address space. For efficiency reasons, we pack all migrating individuals in a single message, which minimizes message startup and transmission time.

After code parallelization, suitable values for a number of parameters of the distributed algorithm must be chosen. Besides the usual global population GP values for each subpopulation one has to define the topology and the number of the communicating subpopulations, the size N of the subpopulations, the migration frequency M, the number of migrating individuals K and the individual replacement policy. We found suitable values by trial and error, running many times the parallel algorithm on the well-understood multiplexor problem [6]. The size of the subpopulations for the present problem was thus set at 400, migration took place every 7 generations and the number of individuals exchanged was 8 to 10% of the subpopulation size. These values were close to those used in [9] and in [7].

We experimented with only one processor topology: the ring, and we tested two synchronous exchange policies: simply passing individuals to the next island in the ring, alternating directions at each swap, and passing individuals "modulo" the swap number (fig. 3). The "modulo" swap gave the best results and we retained it for the subsequent tests. We choose to migrate the best K individuals from each island.

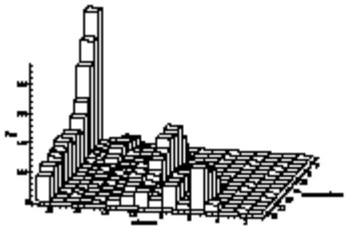

Figure 3

go to the summary

The replacement policy used was that the new K individuals displace the worst K individuals of the receiving population. Here also other alternatives are possible. A GP run is terminated either by finding a solution (i.e. a 0 fitness individual) in any subpopulation or by reaching a maximum number of generations. The following pseudo-code gives a schematic description of the algorithm:

```
initialize P subpopulations  of size N each
generation number := 1
while termination condition not met  do
   for each subpopulation do in parallel
      evaluate and select individuals by fitness
      if generation number mod frequency = 0 then

         send K<N best individuals to a neighbouring subpopulation

         receive K individuals from a neighbouring population

         replace K individuals in the subpopulation

      end if
         produce new individuals by crossover
   end parallel do
   generation number := generation number + 1
end while
```

go to the summary

Results and Conclusions

We did many parallel GP runs for each of the two choices of functions and terminals described in the previous section. The parallel GP based on the first terminal and function set never found a solution in the allowed maximum number of generations (60). A typical run is shown in Fig. 4, in which the number of individuals having a given fitness value generation by generation is presented. It is clear that there is little improvement after about 40 generations and the search stagnates. The best-of-all-runs individual attained a fitness value of 2. The sequential program was never able to find a fitness value better than 4 even when given a maximum number of 100 generations. Clearly, the reason for the unsatisfactory performance both of the sequential and parallel algorithms was the presence of a large percentage of invalid diagrams in the populations. This is so in spite of the worst possible fitness value given to them because crossover, being unrestricted, continuously produces invalid diagrams.

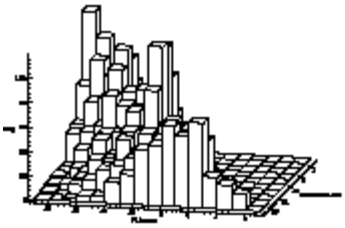

Figure 4

The second terminal and function set choice proved to be much more adequate. The parallel GP was able to find the correct solution on most of the 30 runs. A typical successful run is graphically depicted in fig. 5. Comparing it with fig. 4, it is seen that the average fitness is much lower and that the run quickly converges to a 0 fitness solution.

Not only did the parallel algorithm perform much better from the point of view of computing times, which was obviously expected, it also converged more often to the correct solution than the sequential one using a smaller number of fitness evaluations i.e., with a reduced computational effort. For instance, a particular run on a 8-processor system took 33 seconds to complete successfully with a population of 400 individuals per processor whereas for the same total population size (i.e., 400 x 8 = 3200) a sequential execution on one processor took approximately 10 minutes to complete. We observed the same effects in parallel GA systems [9] and the detailed results reported in [7] also agree with this general trend.

It is difficult in the present case to generalize this results. Indeed, we observed that the speedup fluctuates if processors are added to the system. We think that this phenomenon is due to the particular problem treated here, which seems to be not hard enough to require all that parallel processing power. In fact, with more processors one could use larger populations. But larger populations are not needed in our case and subdividing a relatively small population on more processors increases communication overheads and diminishes subpopulation diversity. Preliminary results on a more difficult problem in evolving financial trading models with a parallel GP system showed a more consistent behaviour with nearly linear speedups (work in preparation). Actually, we found that this last problem is only tractable within reasonable time limits by using parallel GP.

In conclusion, in this work we implemented a parallel GP programming model and we did preliminary experimentation with binary decision diagrams optimization problems. Although parallelism was moderately beneficial in this particular case, more complex problems will benefit even more, allowing larger populations and more difficult fitness functions to be treated. The work of others [7] and our own current work in this direction is promising.

go to the summary

References

[1] Actel. FPGA Data book and design guide. Sunnyvale, Calif., 1995.

[2] S. Arnone, M. Dell'Orto, A. Tettamanzi and M. Tomassini, *Highly Parallel Evolutionary Algorithms for Global Optimization, Symbolic Inference and Non-Linear Regression*, Proceedings of the 6th International Conference on Physics Computing, European Physical Society, Geneva, 51-54, 1994.

[3] E. Cerny, D. Mange and E. Sanchez, *Synthesis of minimal binary decision trees*, IEEE Transactions of Computers, 28, 472-482, 1979.

[4] T3D Software Overview Technical Note, SN-2505 1.1, Cray Research Inc., 1993.

[5] V.S. Gordon and D. Whitley, *A Machine-Independent Analysis of Parallel genetic Algorithms*, Complex Systems, 8, 181-214, 1994.

[6] J.R. Koza, *Genetic Programming*, MIT Press, Cambridge, MA, 1992.

[7] J.R. Koza and D. Andre, *Parallel Genetic Programming on a Network of Transputers*, Computer Science Department, Stanford University, Technical Report CS-TR-95-1542, 1995.

[8] C. Y. Lee, *Representation of switching circuits by binary-decision programs*, Bell Syst. Tech. J., 38, 985-999, 1959.

[9] A. Loraschi, A. Tettamanzi, M. Tomassini and P. Verda, *Distributed Genetic Algorithms with an Application to Portfolio Selection Problems*, in Proceedings of the Int. Conf. on Artificial Neural Nets and Genetic Algorithms, D.W. Pearson, N.C. Steele and R.F. Albrecht (Editors), Springer-Verlag, 384-387, 1995.

[10] P. Marchal and A. Stauffer, *Binary decision diagram oriented FPGAs* in Proceedings of FPGA'94, 2nd International ACM/SIGDA Workshop on FPGAs, Berkeley, Calif., Feb. 13-15, 1-10, 1994.

[11] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*, Springer-Verlag, Second Edition, Berlin, 1994.

[12] B. M. E. Moret, *Decision trees and diagrams*, Computing Surveys 14, 593-623, 1982.

[13] J. Rose, A. El Gamal and A. Sangiovanni-Vincentelli, *Architecture of PPGAs*, Proceedings of the IEEE, 81, 1013-1029, 1993.

[14] W.A. Tackett and A. Carmi, *Simple Genetic Programming in C*, available through the genetic programming archive at ftp.io.com/pub/genetic-programming /code/sgpc1.1.tar.Z.

[15] M. Tomassini, *A Survey of Genetic Algorithms*, to appear in Annual Reviews of Computational Physics Vol. III, D. Stauffer Editor, World Scientific, 1996.

go to the summary

©

REFER TO CONTENTS

Comments

abstract
in french