A TOTALLY DISTRIBUTED GENETIC ALGORITHM: FROM A CELLULAR SYSTEM TO THE MESH OF PROCESSORS

Lukáš Sekanina and Václav Dvo□ák
Department of Computer Science and Engineering
Brno University of Technology
Bo□et□chova 2, 612 66 Brno,
Czech Republic
E-mail: sekanina@dcse.fee.vutbr.cz

KEYWORDS
Distributed processors, simulators, special-purpose processors, performance analysis.

ABSTRACT

The paper deals with properties of the totally distributed genetic algorithm RGA, initially designed for the PIG cellular system. We have adopted the algorithm to a mesh of processors. Simulations were performed using Transim tool in order to investigate performance of this new RGA algorithm independently of a given application. This way, characteristics such as efficiency, speedup, communication delays, the influence of chromosome length and fitness calculations are easily evaluated beforehand.

INTRODUCTION

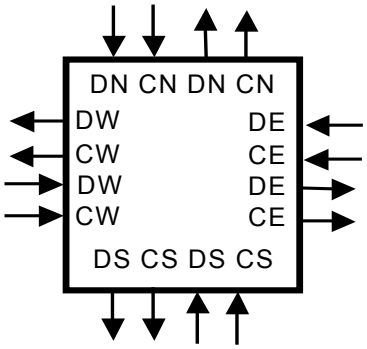
The biological principles like evolution or development, known from nature, have not been only applied at software level, but in last years also directly in hardware. Thus new approaches such as evolvable hardware (Sanchez and Tomassini 1996), cellular programming (Sipper 1997) or embryonic electronics (Mange et al. 2000) have been introduced. The PIG is one of the newest cellular architectures. Typical features and problems of such architectures are discussed in (Sekanina and Drábek 2000). The RGA is a parallel variant of genetic algorithm, developed for the PIG. The RGA is completely implemented using the cells of the PIG and, furthermore, the cells are the subjects of evolution.
We have adopted this new "micro-world" genetic algorithm to the "macro-world" of the common processors. The *Transim* simulation tool was used to investigate the system performance. Simulations allowed us to uncover properties of the RGA for a given system parameters (as a processor frequency, communication delays or chromosome length). The proposed simulation model can be used for decision about usability of the RGA for a given application.

THE PIG ARCHITECTURE

Quite a new computational architecture – *Processing Integrated Grid PIG* (US patent #5886537) – was initially presented at the *First NASA/DoD Workshop on Evolvable Hardware* in 1999 (Macias 1999a). Its idea and features resounded in many other papers, e.g. (de Garis 1999, Miller 2000). The PIG is a large two-dimensional array of identical configurable cells. In future, the cellular array should consist of billions of cells. Each cell communicates locally only with its four neighbors, see Figure. 1. Similarly to a cellular automaton, the computation is done synchronously, in parallel and in discrete time steps. Two neighbors are connected using two (input and output) data wires and two (input and output) control wires.

In *data mode* (i.e. all control inputs are deactivated), the cell operates as a pure combinational circuit, which reads its four D-inputs, uses them as an index to its truth table (to select one of 16 rows) and determines a set of eight output values to be presented on the four D- and four C-outputs. In the *control mode* (i.e. when at least one control input is activated), D-inputs are serially shifted into the cell's internal truth table, according to system-wide clock. This allows one cell to write another cell's internal truth table, which subsequently affects that cell's behavior when it returns to the data mode. Additionally, as the new truth table is shifted into the cell, the cell's prior truth table is shifted out on its D-outputs, and is available for reading. This way, a cell can configure any other cell in the array.

The PIG is infinitely scalable, massively parallel, self-configurable architecture with features of a *dataflow* machine. Because of the PIG's distributed configuration control, it can be used to study not only parallel execution of algorithms in hardware, but also parallel reconfiguration of hardware. Moreover, the PIG can implement circuits, which create new circuits, which themselves create and modify other circuits. But, how shall a designer find the tables of the cells? Due to massive parallelism, the traditional programming approach fails. An *evolutionary algorithm* is usually used to find tables of the cells for a given task (Sipper 1997). Some evolutionary designs (as replicators, counters, and guards) have been successfully created and then stored in libraries to be reused in other projects (Macias 1999b). The next section summarizes typical parallel implementations of the genetic algorithm and explains how the RGA is employed in the PIG.



(A)

inputs				outputs							
DN	DS	DW	DE	CN	CS	CW	CE	DN	DS	DW	DE
0	0	0	0	shiftable combinational							
.	.	.	.	table							
1	1	1	1								

(B)

Figure 1. A diagram of a single PIG's cells: (A) One control **C** and one data **D** bit are read from each neighbor of the cell. One control and one data bit are also sent to each neighbor. (B) Every cell operates according to its own table. In the data mode, the outputs are given by a row of the table determined by the actual combination of D inputs. In the control mode, the table is serially shifted in every system-wide clock.

PARALLEL GENETIC ALGORITHMS AND RGA

Parallel implementations of evolutionary algorithms are summarized in (Cantú-Paz 1999, Tomassini 1999). The following approaches are usually used: (1) Parallel execution of fitness calculation (individuals are divided among processors), (2) the same algorithm is executed on many processors and the best solution is considered, (3) the population is divided into subpopulations (demes), which evolve concurrently and from time to time exchange the best genetic material and (4) individuals are collected in the mesh and only local interaction during evolution (genetic operations) is permitted.

The *RGA* (*Ringed Genetic Algorithm*) (Macias 1999b) is a kind of parallel genetic algorithm of the class (4), which was implemented in the PIG. The standard genetic algorithm is not scalable well, since the execution time depends linearly on the size of population. In case of total hardware implementation, all individuals have to be presented in hardware and, therefore, they could be evaluated in parallel. It means that the logic for fitness calculation and genetic operations have to be distributed along individuals. Truth tables of several cells are evolving subjects in the case of the RGA in the PIG. Thus, distributed reconfiguration of hardware has to be requested, too. The PIG is able to ensure all these requirements and the calculation of the new population at constant time complexity.

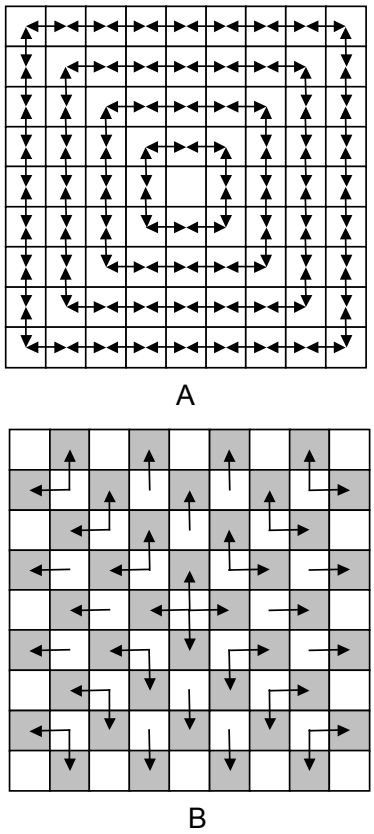


Figure 2. Phases of the RGA. The arrows show the communication partners and a direction of communication for every cell: (A) *Local communication on the circle.* A central individual is not used. (B) *Communication between circles.* The white-marked individuals (writers) rewrite genetic information of the gray-marked individuals (readers).

Figure 2 shows a population of 9 x 9 individuals organized in a mesh. An individual (a single square in the Figure 2) consists of several cells (whose tables are subjects to evolution), a logic for communication with neighbors, a logic for fitness calculation and a logic for execution of genetic operations (crossover and mutation). All that is implemented using only the cell as a building block. A genetic algorithm is distributed totally – individuals operate autonomously and concurrently. The shifting of the truth tables allows entire groups of cells to "travel" in the mesh and to exchange genetic information among neighbors. Fitness calculations as well as genetic operations are performed in parallel – independently of the size of population. Only the simulation model of RGA (written in C++) exists nowadays. The evolution is executed in two phases, until the 100%-quality solution appears:

Phase #1: The individuals communicate on circles around and except the central individual. When the first population is generated (randomly, in parallel), the fitness values are calculated. A repetition of the genetic operations among the neighbors on the same circle (with consequent fitness calculations) will ensure that several high-quality individuals will travel around circles. The crossover is applied only to individuals with similar quality. In case of great differences, a worse-quality individual is replaced by a better one. The number of these interactions is fixed.

Phase #2: The genetic information is exchanged between circles in parallel and in one step. Individuals are divided into *readers* and *writers*. A writer will use its own genetic information to replace reader's genetic information (Figure 2b). The central individual operates as a writer and sends randomly generated genetic information to its four neighbors to ensure diversity of population.

THE RGA IN THE MESH OF PROCESSORS

Inspired by the previous architecture, we applied an idea of RGA to the mesh of common processors. The processor manages just one individual, calculates its fitness, communicates with neighbors and performs local genetic operations. In other words, a single processor simulates a group of the cells and communication cables between processors simulate another group of PIG's cells! We do not investigate the efficiency of evolution for a single particular problem, but efficiency of architecture and communication for a class of applications.

To estimate system performance, *Transim* simulation tool was used. *Transim* (Hart 1993) is a case tool intended to give guidance on performance issues early in the design stages of a parallel processing project, to allow rapid prototyping and conceptualization. The simulation model in *Transim* is written in the subset of *Occam* language. *Transim* can simulate a single processor or a complex network. It can simulate a single process or a large application in which many individual processes are to be active on each processor (a processor is often termed a 'node' in *Transim* parlance). Parallel execution, alternation, channel communication, time-slicing, priorities, interruption, concurrent operation of links and the effects of external memory are taken into account.

A mesh topology is modeled. Six channels (two for intra-ring and four for inter-ring communications) are associated with each of N x N processors (only an odd N and N □ 3 may be used). In the initialization phase, based on the spatial location, the processor determines its neighbors in the ring, neighbors for the inter-ring communication and its role (writer, reader, central). Four inter-ring channels are not used effectively, but on the other hand, the simulation model is transparent. These four channels are used only by the central processor, two channels are used by diagonal processors, one channel by other writers while readers will not use any of them. This incongruity brings some difficulty to simulation. Procedure *SERV*, which is activated in case of useful processor's operation, simulates the time of fitness calculation and genetic operations inside the ring. Communication time depends on the number of transferred bytes of fitness values and neighbor's genetic information. The phase #2 is only one parallel communication in which reader's genetic information is replaced. The source code of the simulation model is available on request.

PERFORMED SIMULATIONS

The goal is to investigate large space of possible applications for the RGA adopted in common processors and to determine a class of applications suitable for RGA. Table 1 summarizes main parameters used in simulations. Many experiments have been performed – the following Figures 3a-d show how efficiency depends on a selected parameter while the other parameters remain constant and set up according to Table 1.

Table 1: Parameters of the simulation model

Value	Description
20MHz	Processor's frequency
20Mbit/s	External channel speed
9x9=81	Processors in a mesh
36	A number of intra-ring communications
32	Chromosome length in bytes
1024	CPU's clocks for fitness calculation
2	Byte count of fitness
100	CPU's clocks for genetic operations

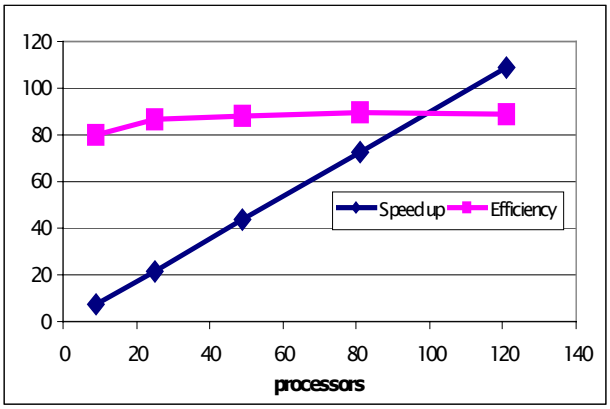


Figure 3a. The dependence of speed up and the efficiency on the number of used processors. N x N processors are used in the mesh. N = 3, 5, 7, 9, 11 (for values in Table 1).

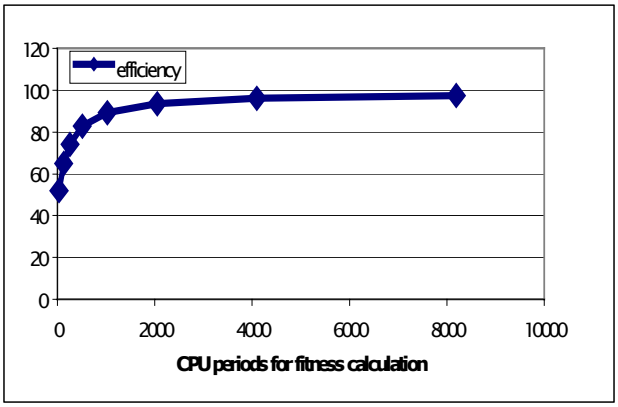


Figure 3b. The dependence of efficiency on the time of fitness calculation (for values in Table 1).

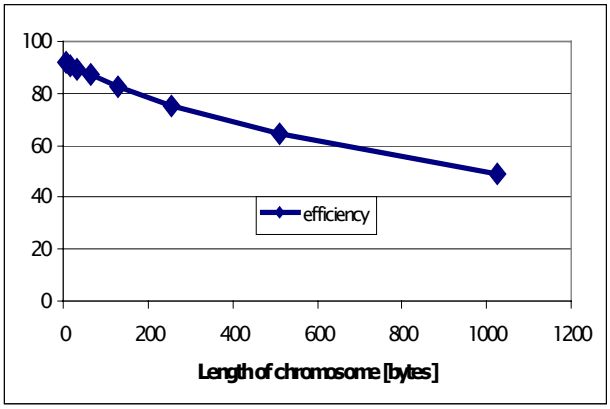


Figure 3c. The dependence of the efficiency on the chromosome length (for values in Table 1).

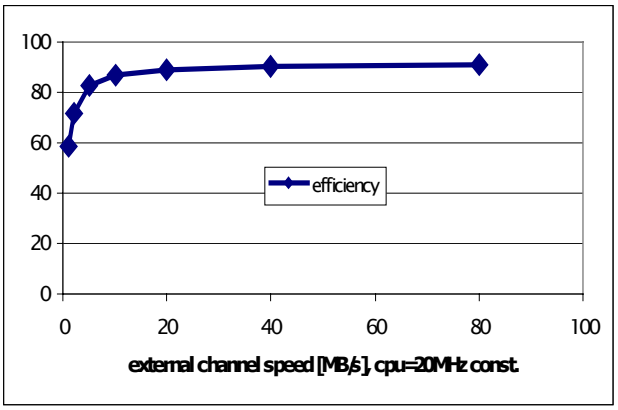


Figure 3d. The dependence of the efficiency on the external channel speed (for values in Table 1).

RESULTS

Experiments show that:

- Efficiency is constant, independently of the population size (i.e. the number of processors used). The problem is excellently scalable. Only the central processor in the phase #1 and writing processors at the boundary (in the phase #2) are not used. In practical applications, the central processor can be omitted and its neighbors will fulfill its role (of a random chromosome generator).
- The RGA will be excellent in applications with very time consuming fitness calculation (e.g. in the field of evolvable hardware). In case of simple fitness calculation, the communication overhead will be dominant. Communications can not be overlapped by some useful calculations.
- Increasing length of chromosomes leads to decreasing of efficiency.
- Optimal efficiency was reached in case of at least the same speed of communication lines [Mbit/s] and processors [MHz].
- An unconventional processor count in the RGA can be sometimes disadvantageous.

CONCLUSIONS

We have applied unconventional kind of parallel genetic algorithms to the mesh of processors and investigated system efficiency. The main problem was to simulate irregularity of communications. The proposed model allows realistic performance prediction for a given application and system parameters. The user can easily estimate application performance and thus decide on benefits of RGA for a given application. The class of RGA applications mainly includes such applications where the fitness calculation is the most time consuming operation. We are going to use RGA for some evolvable hardware based applications in future.

REFERENCES

Cantú-Paz, E. 1999 "Designing Efficient and Accurate Parallel Genetic Algorithm", PhD theses, University of Illinois.

de Garis, H. 1999. "Review of Proceedings of the First NASA/DoD Workshop on Evolvable Hardware." *IEEE Transactions on Evolutionary Computation*, Vol. 3, No. 4, (Nov), 304–306.

Hart, E. 1993. *Transim User Guide and Reference Manual*, Version 3.5, University of Westminster.

Macias, N. 1999a. "The PIG Paradigm: The Design and Use of a Massively Parallel Fine-Grained Self-Reconfigurable Infinitely Scalable Architecture." In *Proc. of The First NASA/DoD Workshop on Evolvable Hardware (EH'99)*, Stoica, A., Keymeulen, D., Lohn, J. (Eds.). IEEE Computer Society, Pasadena, California, 175–180.

Macias, N. 1999b. "Ring Around the PIG: A Parallel GA with Only Local Interactions Coupled with a Self-Reconfigurable Hardware Platform to Implement an O(1) Evolutionary Cycle for EHW", Internet presentation of Cell Matrix Corporation,

URL:
http://www.cellmatrix.com/entryway/products/pub/publications.html

Mange, D. et al. 2000. "Towards Robust Integrated Circuits: The Embryonic Approach." *Proceedings of the IEEE*, Vol. 88, No. 4, (Apr), 516 – 541.

Miller, J. F. 2000. "Review of the 1st NASA/DoD Workshop on Evolvable Hardware 1999". *Genetic Programming and Evolvable Machines*, Kluwer Academic Publishers, Manufactured, Vol. 1, No. 1, 171–174.

Sanchez, E., Tomassini, M. 1996. "Towards Evolvable Hardware: The Evolutionary Engineering Approach", LCNS 1062, Springer-Verlag Berlin Heidelberg.

Sekanina, L., Drábek, V. 2000. "Relation Between Fault Tolerance and Reconfiguration in Cellular Systems." In *6th IEEE Int. On-Line Testing Workshop*, Palma de Mallorca, Spain, 25 – 30.

Sipper, M. 1997. *Evolution of Parallel Cellular Machines: The Cellular Programming Approach*. Springer-Verlag Berlin Heidelberg.

Tomassini, M. 1999. "Parallel and Distributed Evolutionary Algorithms: A Review" In: *Evolutionary Algorithms in Engineering and Computer Science*, K. Miettinen, M. Mäkelä, P. Neittaanmäki and J. Periaux (eds), J. Wiley and Sons, Chichester, 113–133.

ACKNOWLEDGEMENT

This research has been carried out under the financial support of the Research intention no. CEZ: J22/98: 262200012 - Research in information and control systems.

BIOGRAPHY

LUKÁŠ SEKANINA received MSc degree in Computer Science and Engineering from Brno University of Technology, Czech Rep. in 1999. He was visiting lecturer at Pennsylvania State University, USA in the spring semester 2001. Currently he is a PhD candidate at the Brno University of Technology. His research interests focus on evolvable hardware and cellular computational architectures.
Email: sekanina@dcse.fee.vutbr.cz
WWW: http://www.fee.vutbr.cz/~sekanina

VÁCLAV DVORÁK received MSc and PhD degrees in Electrical Engineering from Brno University of Technology in 1963 and 1968. He was with Res. Inst. of Mathematical Machines in Prague till 1973, and then Assoc. and Full Prof. at Brno Univ. of Technology. Until now, he spent also over 8 years with universities abroad (Canada, Malta, Libya, New Zealand, Australia). His recent research concentrated on advanced computer architecture and parallel and distributed computing.
Email: dvorak@dcse.fee.vutbr.cz
WWW:http://www.fee.vutbr.cz/~dvorak