GRID COMPUTING IN MATLAB FOR SOLVING EVOLUTIONARY ALGORITHMS

M. Linder, D.Pernecký, I. Sekaj

Institute of Control and Industrial Informatics
Faculty of Electrical Engineering and Information Technology
Slovak University of Technology in Bratislava, Slovak Republic

Abstract

Evolutionary algorithms (EA) represent a powerful optimization approach, which mimics the process of natural evolution. These algorithms are successful in solving complex problems. But in practical applications the computation time also under the use of such algorithms can be very long. In this paper we present a grid solution for evolutionary algorithms running in Matlab, where the solving time is decreasing thanks the distribution of the computation process into a many processor/PC structure.

1 Introduction

In general EA are meta-heuristic algorithms, which mimics the powerful process of natural evolution. They operate over a set of potential solutions (population) in each computation cycle. They use simple operations like selection (surviving of best individuals), crossover (combination of parent genes) and mutation (modification of parent genes) using which they are able to search for (optimize) good or even the best solutions in the given search space. The kernel of the optimization process is the success measure (criterion function) of each individual - "fitness", which is to be minimized or maximized respectively. Evaluation of the fitness depends on the given problem, it can contain complex calculations, model simulations etc. and it can take time from fractions of second to tens of seconds, minutes and sometimes also more. Because EA needs normally thousands or hundreds of thousands fitness evaluations, the computation effort is very high. There are several ways how to decrease the computation time of the evolutionary algorithm. The most used way is to distribute the computation process into more computational units. There are various possible architectures which can structuralize population of the EA into the existing computation nodes. For example splitting of the population to low number of relatively big isolated islands - subpopulations (island-based parallel EA) with infrequent communication between these islands (migrations) or with overlapping areas or splitting the population into big number of small islands (it is common to have only a single individual in each island) with frequent communication between them (cellular parallel EA) (Fig.1) [4]. Such parallelization leads to decreasing number of computation cycles – "generations" and also to better solutions. There are several approaches how to do it. The first approach is to locate each subpopulation into its own computational unit. This corresponds to the coarse-grained parallel EA. Another approach is to have a master-slave structure. In such a structure the master obtains all individuals of all islands, computes the EA operations (selection, mutation, crossover, migration) excluding the fitness computation, which only is distributed into the slaves - other computation nodes (processors/PCs).

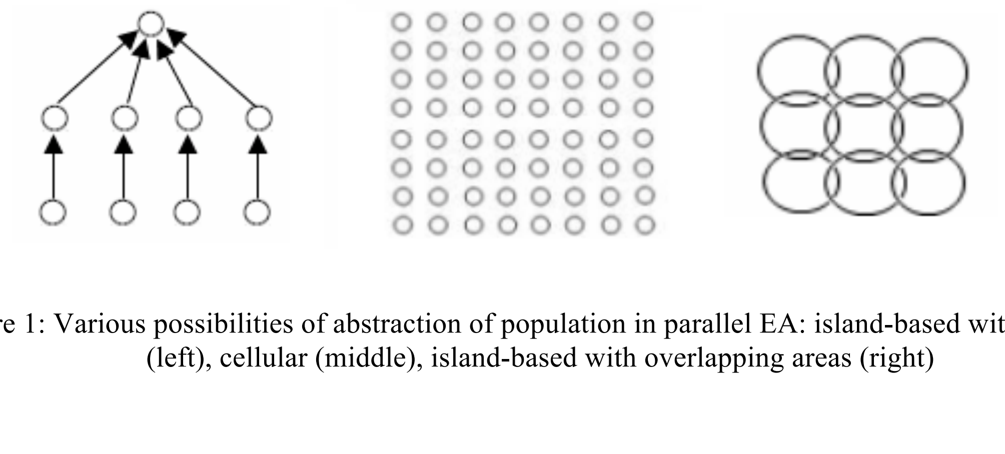


Figure 1: Various possibilities of abstraction of population in parallel EA: island-based with migration (left), cellular (middle), island-based with overlapping areas (right)

Our goal was to design and to implement a general computational platform for EA and parallel EA (PEA) in the Matlab environment, which is independent on the structure of the subpopulation organization. We proposed a master-slave architecture (Fig.2), where the master executes the evolutionary algorithm and the clients (slaves) only compute the fitness. Such architecture allows to organize the PEA in arbitrary structures (island-based, cellular, etc.) and to distribute the time-consuming fitness evaluations into more computation nodes. The communication layer between clients and the master is provided by the Microsoft SQL database and FTP server. The SQL database is able to decrease the communication load between the master and slaves, it is very scalable and intended for communication with a large number of clients, it solves the communication conflicts and it is capable to process a big amount of data in relatively short time. For increasing the processing speed we used the RAM disk for locating the database physical files, where the R/W operations are much faster. This solution is similar to In-memory database. For computation speed comparison the CrystalDiskMark software have been used. Following test have been compared: Seq - Sequential Read/Write Test (Block Size = 1024KB), 512K - Random Read/Write Test (Block Size = 512KB), 4K - Random Read/Write Test (Block Size = 4KB), 4K QD32 - Random Read/Write Test (Block Size = 4KB, Queue Depth = 32) for NCQ&AHCI [3]. The results are shown in Fig.3.

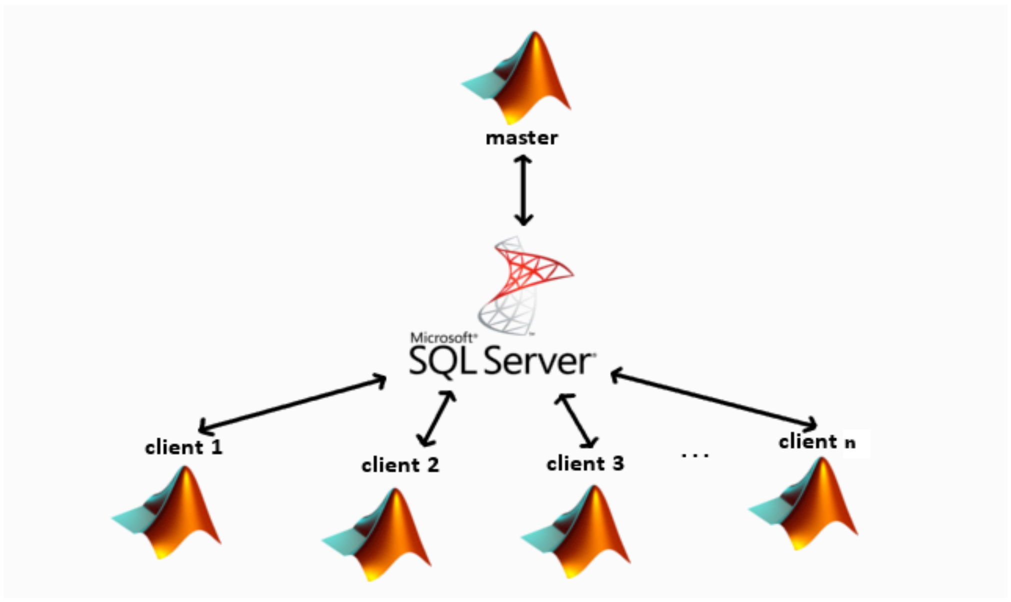
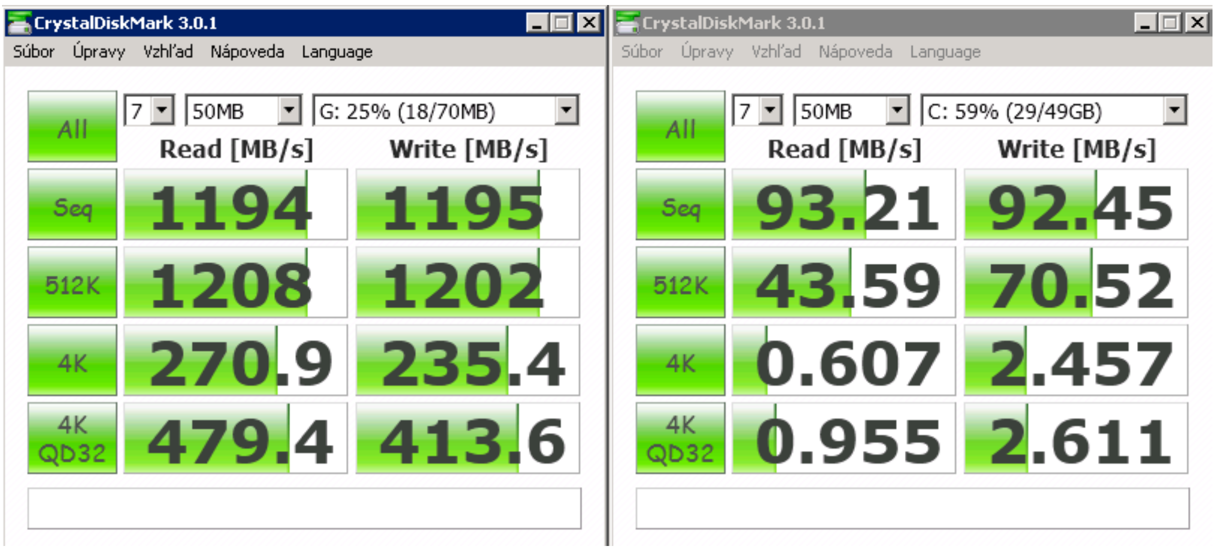


Figure 2: GRID computing architecture


CrystalDiskMark 3.0.1
Súbor Úpravy Vzhľad Nápoveda Language
All
CrystalDiskMark 3.0.1
Súbor Úpravy Vzhľad Nápoveda Language
All

Left: 7 | 50MB | G: 25% (18/70MB)

	Read [MB/s]	Write [MB/s]
Seq	1194	1195
512K	1208	1202
4K	270.9	235.4
4K QD32	479.4	413.6

Right: 7 | 50MB | C: 59% (29/49GB)

	Read [MB/s]	Write [MB/s]
Seq	93.21	92.45
512K	43.59	70.52
4K	0.607	2.457
4K QD32	0.955	2.611

Figure 3: Comparison of performance of RAM disk (left) and classical harddrive (right)

2 Technical realization of the GRID

There are more possibilities how to connect Matlab with the database. One of them and probably the most used is the Matlab built-in Database toolbox, which is multiplatform. But we preferred Microsoft OLE connector, because it is faster than the Database toolbox. The difference between these two alternatives is shown in Tab. 1.

Table 1: COMPARISON OF 1000 RUNS OF THE GRID DATABASE FUNCTIONS ON LOCALHOST

		Processing time in seconds
Microsoft OLE	Update of 50 individuals (individual size is 3 genes)	17.6068
	Update of 1000 individuals (200 genes per individual)	3.0848e+003
	Fitness writing of 1 individual (3genes per individual)	0.342
	Reading fitness of 50 individuals (3 genes per individual)	5.3801
Matlab Database toolbox	Update of 50 individuals (individual size is 3 genes)	116.6693
	Update of 1000 individuals (200 genes per individual)	5.0433e+003
	Fitness writing of 1 individual (3genes per individual)	0.532
	Reading fitness of 50 individuals (3 genes per individual)	9.6055

The initialization phase of the GRID computing contains following steps:

1. At the beginning the master and clients (slaves) make a connection to the database.
2. Master creates all tables necessary for the computing in SQL database: table for service data (service table) and table where individuals and their fitnesses are stored (working table).
3. Master uploads all necessary files to the FTP server and writes to the service table the name of the executable file (name of function which evaluates fitness of given individuals). Master inserts the new population to the working table and sets their fitness to zero and semaphores to zero. Then it starts the solving by changing run flag in the service table to 1.
4. Clients periodically check these run flags in the service table and if this flag changes to 1, clients are ready to start computing. Master is now waiting for clients to finish the evaluation of individuals in the working table.
5. Clients download files from FTP server. The names of the executable files form the service table are transformed to executable functions. Each client demands the database sequencer for a unique ID, which is needed for identify themselves in the working table. In this state the initialization of the clients is finished and the clients are ready for computing the fitness.

The processing of the GRID consists from following steps:

1. Clients start taking the given number of individuals from the table and evaluate their fitness in the cycle. The process of taking individuals from the working table uses additional semaphores (each individual in the working table has its own semaphore variable). If the semaphore is set to zero it means the individual is not evaluated, if the semaphore value is 1, the individual is evaluated, if semaphore value is 2 we need not to evaluate the individual (because of some aspects of EA).

2. When all individuals are evaluated, the master pauses clients by changing value of the run flag in service table to the value 2. The master continues with next EA steps until all individuals of all populations are updated in the database. Master in the first step starts clients and then update the working table with new individuals. Starting clients before updating saves time. In the same time as master is updating the table, clients could evaluate the first individuals. Updating of the table is a more time consuming operation then the selection of few individuals.
3. The master is waiting for fitness calculation of the clients. If some client does not response (this could be caused by software, hardware or network problems or thanks some other unexpected events), the master changes its semaphore value to zero and this individual is evaluated by another client.
4. If the complete computation cycle of the EA is finished, the master stops all clients.

3 Experimental results

For testing the proposed parallel computation architecture the following hardware has been used: master and clients were run on a machine with 2 Quad core processors AMD Opteron 2354 with 16 GB RAM (8 cores), the SQL database was on a PC with an AMD Athlon 6000+ processor with 2GB RAM.

To make sure that the Master-slave architecture has better performance we must keep the following relationship [1]:

$$P = \frac{T_s}{T_p} > 1 \tag{1}$$

where *P* is the performance factor T_s is the time of evaluation of the EA algorithm, which runs on a single core and T_p is the time of the EA evaluation for the parallel configuration. A genetic algorithm with 50 individuals in population has been considered (but only 38 fitness values were evaluated in each generation, the rest are unchanged individuals with known fitness from previous generation). The number of generations was set to 1000. Number of clients (slaves) are 6. In our case the performance factor for the 1st application problem, where the fitness evaluation of a single individual takes 0.2293 seconds, is *P=5,291* and for the 2nd problem, with fitness evaluation time 2.2101 seconds, *P=5,432*. In the ideal case should be *P=6* (equal to number of processors), but the real number is lower because of communication time. The efficiency of parallel implementation we can define as

$$Efficiency = \frac{T_s}{T_p N} \times 100[\%] \tag{2}$$

where *N* is number of computing cores. The efficiency for the 1st problem is 88,194 % and for the 2nd problem it is 90,537%.

In a next test the following configuration has been used: Intel Core i7 860 processor with 4 GB RAM. We have used 4 clients, the configuration of the genetic algorithm was: 50 individuals in population (but only 38 were evaluated each generation). The number of generations was set to 100. The results are shown in Tab. 2.

Table 2: EFFICIENCY OF THE GRID

Evaluation of a single individual [s]	T_s [s]	T_p [s]	Efficiency [%]
0,1091493	414,9744387	119,5956316	86,74
0,2028549	770,6914000	218,1047329	88,33
1,0140683	3853,2000000	1021,3769092	94,31
5,0076530	19029,0000000	5017,0713304	94,82

In case when $T_n << T_{fit}$ for N computing units:

$$\lim_{T_{fit}\to\infty} \frac{T_s}{T_p} = \lim_{T_{fit}\to\infty} \frac{T_{GA} + T_{fit}}{T_{GA} + \frac{T_{fit}}{N} + T_n} = N \tag{3}$$

where T_{GA} is time of genetic algorithm without time of evaluation of fitness, T_{fit} is time of evaluation of fitness and $T_n = T_{net} + T_{SQL}$, T_{net} is time-delay of the Ethernet network and T_{SQL} is time-delay of the SQL server.

4 Conclusions

In this paper we presented a grid solution for EA and PEA in the Matlab environment. It is built on the master-slave architecture and it was developed using Matlab and MS SQL. This approach opens a way for solving time consuming applications using a wide range of parallel computing architectures (single EA, coarse-grained PEA, fine-grained PEA or hybrid PEA). Our solution is suitable for EA/PEA where evaluation of fitness needs relatively long time (complex simulations, construction tasks, etc.). Using this grid we are able to reduce the computation time of complex practical applications in Matlab from days to hours, from hours to minutes.

Acknowledgement

This work has been supported from the grant of the Slovak Grant Agency VEGA No. 1/0690/09.

References

[1] E. Cantú-Paz, D. E. Goldberg: *Efficient Parallel Genetic Algorithms: Theory and Practice*. Computer Methods in Applied Mechanics and Engineering, 2000.

[2] E. Alba, M. Tomassini, *Parallelism and Evolutionary Algorithms*. IEEE Trans. On Evolutionary Computation, Vol. 6, NO.5, October 2002

[3] CrystalDiskMark benchmark software, Available at http://crystalmark.info [11.10.2012]

[4] I. Sekaj, M. Linder, D. Pernecký: *Experimental comparison of selected types of parallel evolutionary algorithms*, IJCCI 2011, October 2011, Paris

Marek Linder
marek.linder@stuba.sk

Daniel Pernecký
xperneckyd@stuba.sk

Ivan Sekaj
ivan.sekaj@stuba.sk