Integrated Software Platform for Parallel Global Optimization

Ewa Niewiadomska-Szynkiewicz[1,2] and Agnieszka Zawadzka[1]

[1] Warsaw University of Technology, Institute of Control and Computation Engineering, Warsaw, Poland,
[2] Research and Academic Computer Network (NASK), Warsaw, Poland
e-mail: ens@ia.pw.edu.pl, a.zawadzka@elka.pw.edu.pl

Abstract. The paper[1] is concerned with global optimization techniques and their parallel implementation. We describe an integrated software platform GOOL–PV (Global Optimization Object-oriented Library–Parallel Version) that provides the tools for solving complex optimization problems on parallel and multi-core computers or computer clusters. Finally, we present the comparative study of sequential and parallel global optimization algorithms based on numerical results for a standard set of multimodal functions.

1 Introduction

Many engineering problems are formulated as the optimization tasks in which the objective function is not convex and posses many local minima in the region of interest. In addition, in many practical contexts, the optimization problem cannot be described analytically due to the natural complexity and uncertainty of the real-life systems. In such cases the simulation experiment is usually used to evaluate the expected performance of the system for each set of decision variables. It involves of simulation-based optimization (or simulation optimization) that is the merging of optimization and simulation techniques. The usage of traditional, local optimization methods is usually inefficient for solving multimodal or simulation-based problems. Therefore, methods designed for global optimization are important from a practical point of view.

Global optimization is generally complex and usually involves cumbersome calculations, especially when consider simulation-optimization case when we have to perform simulation experiment in every iteration of the algorithm. The restrictions are caused by demands on computer resources – CPU and memory. The directions, which should bring benefits are:

- hybrid techniques that combines global and local algorithms, to solve the optimization problem,
- parallel computing where the whole task is partitioned between several cores, processors or computers.

Hybrid approaches can speed up the convergence to the solution. Parallel implementation allows to reduce the computation time, improve the accuracy of the solution, and to execute large program which cannot be put on a single processor.

[1] This work is partially supported by Ministry of Science and Higher Education grant N N514 416934.

Recently a number of software packages with numerical solvers for global optimization have been developed, and can be find in the Internet. They support sequential and parallel programming. Publicly available implementations of interval analysis and branch-and-bound schemes are discussed in [5, 19]. The goal of the COCONUT project [3], was to integrate the currently available techniques from mathematical programming, constraint programming, and interval analysis into a single discipline, to get algorithms for global constrained optimization. Solvers implementing various types of techniques for global optimization (deterministic and stochastic), i.e., interval methods, continuous branch and bound, multistart, genetic and evolutionary, tabu search and scatter search are provided in [18].

This paper is organized as follows. In the first section we describe organization, implementation and usage of our software platform GOOL–PV. Next, the numerical algorithms provided in GOOL–PV are presented. The focus is on hybrid techniques and parallel versions of the algorithms. Finally, the comparative study of efficiency of sequential and parallel versions of selected global optimization methods is presented.

2 Description of GOOL-PV System

GOOL–PV (*Global Optimization Object-oriented Library – Parallel Version*) is an integrated software platform that provides tools for solving the following optimization problem:

$$\min_{x \in \Re^n} f(x) \qquad \text{subject to:} \qquad g_i(x) \le 0, \quad i = 1,...,m \tag{1}$$

where f and g_i are real-valued functions. In general the problem (1) is nonconvex.

GOOL–PV is developed based on GOOL system (described in [15]) – the library of solvers for local and global optimization problems. GOOL–PV is more advanced and has wider range of applications. The main objective was to speed up calculations and improve the accuracy of the solution, so the focus was on parallel computing. The new extended version of GOOL can be used to solve the complex optimization problems on parallel and multi-core machines or computer clusters.

Similarly to the original sequential GOOL system two approaches to user-system interactions, i.e. GOOL–PV/COM and GOOL–PV/GUI are offered. In the GOOL–PV/COM version batch processing is assumed. This type of user-system operation is dedicated to the complex optimization problems, where values of the objective function are calculated based on simulation. GOOL–PV/GUI is dedicated mainly to education and research concerned with testing various algorithms and tuning their parameters. It supplies the graphical environment for optimization problem definition and results presentation. The graphical editor, symbolic expression analyzer and tools for dynamic, on-line monitoring of the calculation results are provided. The following graphical presentation techniques are available: 2D and 3D graphs, leaves of the function values and a table of numbers. In the case of parallel versions of the algorithms the solutions calculated by different processors are presented in the same window but in different colors. The results presentation is fitted to the optimization method (points, lines, grids). The visualization of a multidimension problem is achieved by displaying in the separate windows the leaves for each pair of variables, under the assumption that all other variables are fixed.

2.1 System Architecture and Implementation

The GOOL–PV system is composed of three main components (see Fig. 1):

- *system kernel* that provides runtime infrastructure, manages communication between calculation processes and user interface, and data repository services,
- *graphical user interface* (GUI) that provides a set of tools mainly to support the interaction with the user and the runtime monitoring,
- *library of numerical methods* that provides a set of numerical algorithms divided into two parts: the library of optimization solvers in sequential and parallel versions (GOOL–PV/OM) and the library of random generators (GOOL–PV/RG).

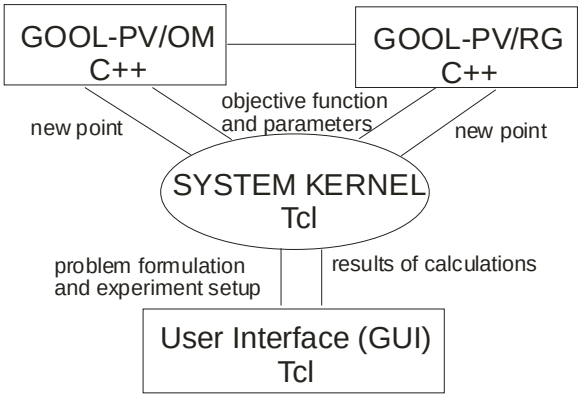

Figure 1. GOOL-PV architecture

GOOL–PV is based on C++ and the script language Tcl (Tool Command Language). All numerical methods are implemented in uniform form as C++ classes while the higher-level activities, i.e., problem definition, parameters setting, results presentation, managing calculations and communication between the optimization engine and the user interface are easily achieved with Tcl. Hence, two functionalities of GOOL–PV are separated and can be easily modified.
The optimization algorithms provided in GOOL–PV/OM library built upon GOOL–PV classes have hierarchical structure. The hierarchy of classes is natural and well defined. Three fundamental classes: *Task*, inserting the considered optimization problem to be solved, *Algorithm*, the basic class of all optimization methods and *Generator*, for random numbers generation are provided. Available software may be easily adopted, new algorithms can be implemented applying classes defined in GOOL–PV. The open design of the system architecture, and its extensibility to include other open source modules, was chosen in the hope that the system will be a useful platform for researchers and students. The code is currently available for MS-Windows and Linux operating systems.

2.2 Library of Numerical Methods

The numerical library consists of two parts GOOL–PV/OM and GOOL–PV/RG containing, respectively a collection of local and global optimization solvers and random numbers generators. The following algorithms for random numbers generation have already been implemented: uniform, normal (Gaussian), Beta, Cauchy and three quasi-random sequences: Halton, Sobol, Fauer [14]. Several techniques for one and multidimensional local and global search are provided. The current version of the system offers global optimization solvers from

two groups: deterministic: chaotic movement [7,16], branch-and-bound and clustering techniques [8, 11], and stochastic: random search (pure and population set based direct search methods) [1], simulated annealing [4], genetic algorithms [6, 17], evolutionary strategies [2, 9, 10] and hybrid methods combining stochastic global search with local deterministic search [12, 20]. All listed algorithms are implemented in a few versions. Most of them can be executed in parallel. The algorithms from the GOOL–PV/OM library can be used to solve unconstrained and constrained optimization problems. The inequality constraints are accounted for in the minimized performance function using simple penalty terms for constraints violation. The detailed description of implemented algorithms can be found in [15].

3 Hybrid Methods

Two hybrid techniques combining well known methods for solving local and global minimum were added to the GOOL-PV/OM library. The goal was to develop algorithms that outperform the simple genetic and simulated annealing algorithms. In this section we provide the short description of these mechanisms.

3.1 Genetic Algorithm with Local Tuning

The genetic algorithm with local tuning called GOD, developed by Yang and Douglas, and presented in [20] integrates the simple genetic algorithm GA described in [6] with the well known downhill method NM developed by Nelder and Mead [14]. The goal was to improve the performance of GA. GOD speeds up the convergence to the global minimum, reduces the demands on the size of the initial population of points, and is more robust w.r.t. the local solutions.

The algorithm operates as follows. In each step a new population is created using both GA and NM methods, i.e. a subset of new points is generated by genetic operations, and a subset by the downhill method. In order to speed up the convergence, the proportion of points generated by both methods varies as the global optimum is approached. The more iterations have been executed, the more points are created using the NM method.

A simple coding is proposed, in which each value is represented by a code value from range (0;1). Crossover is based on random selection of parents' genes, whereas during mutation each gene may be replaced by a random value. In the selection phase the values of fitness calculated for points from old and new population are compared. The points with better fitness are selected.

3.2 Simulated Annealing with Genetic Algorithm

The method called SA/SOS (*Synchronous approach with occasional solution exchanges*) is described in [12]. It combines the simulated annealing SA with the genetic algorithm GA. SA/SOS algorithm operates as follows. Computation is performed on the initial set of points. Simulated annealing is used to transform the population. After assumed number of steps, a crossover operator as implemented in GA is applied to the selected points. The randomly selected coordinates of chosen points are modified. The overall solution quality E is measured as a sum of fitness of all points in the population, $E = \sum_{k=1}^{P} f(x_k)$ where P denotes the size of population of points x_k, $k=1,\ldots,P$, f the performance function (1) (fitness). The control parameter T (temperature) in SA does not change in case when the overall solution E is

improved. Otherwise, it is changed according to the cooling scheme $T = T/(1+\beta E^*/E)$, where $E^* = \sum_{k=1}^{P} f(x_k^*)$, x_k and x_k^* points before and after crossover operation ($k=1,\ldots,P$).

4 Parallel Algorithms

The implementation of parallel versions of optimization algorithms provided in GOOL–PV numerical library is based on the OpenMP tool. The OpenMP (Open Multi-Processing) [13] is an application programming interface (API) that supports multi-platform shared memory multiprocessing programming in C/C++ and Fortran on many architectures. It consists of a set of compiler directives, library routines, and environment variables that influence run-time behavior. An application can run on multiprocessor, multi-core, machines and computer clusters.

The objective of parallel implementations was to speed up the calculations and improve the accuracy of the solution. Several ways of methods parallelization were considered, which resulted several more and less complicated, and at the same time more and less effective variants of each algorithm. In general the hybrid communication model combining master-slave architecture and peer-to-peer communication was applied. We distinguished two groups of threads:

Master (the main thread) initiated by the GOOL-PV systems, which goal is to read all parameters from GOOL-PV/GUI, start the calculations and create slave threads.

Slaves – threads that perform the calculations.

All variants of parallel implementations of given optimization algorithms differ in frequency and organization of slave threads/processes intercommunication. Two following variants are implemented:

A. Several independent instances (slave threads) of a given algorithm are executed for different sets of input data (in case of GA various initial populations), each on a separate processor. The master process is responsible for calculation processes initialization and calculations termination. Each slave stops the calculations after stopping condition is met and sends its results together with the adequate message to the master thread.

B. Similarly to A several independent instances of a given algorithm are executed, each on a separate processor and its own set of input data. The master thread is responsible for calculation processes initialization, master-slave communication and calculations termination. The master-slave communication during the calculation is performed. After each assumed number of iterations each slave sends the message with its local best point/points to the master and takes the best point currently available in the master thread. The master-slave communication is asynchronous. Each slave stops the calculations after stopping condition is met and sends its results together with the adequate message to the master thread.

It should be pointed that different parallel implementation emphasis different aspects, namely a compromise is made between efficiency, accuracy an reliability, where reliability refers to the probability of obtaining a global minimum.

5 Numerical Results

Multiple numerical experiments were performed for four standard multimodal functions defined by Ackley (AC), Levy (LE), Rastrigin (RA) and Griewank (GR), and described in the Appendix. The tests were carried out on the Sun Fire V440 computer equipped with four processors. The objective of the experiments was to compare the performance of the global optimization solvers provided in GOOL–PV. The sequential and parallel versions were considered. The focus was on benefits of parallel implementation. The results of calculations are presented in figures and tables. Three main criteria that determined the performance of the compared algorithms: the quality of the final result, running time and number of function evaluation were taken into consideration.

5.1 Hybrid Methods

The hybrid method GOD was compared with the genetic algorithm GA. The initial population consisted of 40 points in GOD and 160 points in GA. The algorithms were terminated when the difference of performance values for a few trial points was less than ε=0.1.

The SA/SOS performance was compared with the simulated annealing SA. The following cooling schemes for decreasing the control parameter T (temperature) were implemented: in case of SA $T^{i+1} = (1-\alpha)T^i$, where α=0.2, in SA/SOS $T^{i+1} = T^i/(1+\beta E^*/E)$, where β=0.2. The initial population in SA/SOS consisted of 10 points.

Table 1. Comparison of hybrid and traditional methods.

		GOD		GA		SA/SOS		SA	
fun	*n*	*f_min*	*f_eval*	*f_min*	*f_eval*	*f_min*	*f_eval*	*f_min*	*f_eval*
AC	5	0,87	256	7,07	1600160	2,96	130481	3,61	120815
	10	0,93	1654	13,03	1600080	5,18	384525	5,27	362591
	20	3,34	16973	15,86	1600160	6,74	1270940	9,43	1200570
LE	5	0,93	32	1,79	1121440	0,79	2704	7,74	2480
	10	0,92	194	1,36	1487796	18,52	3816	41,59	3639
	20	2,24	18984	32,49	1600160	81,52	12269	178,76	12918
RA	5	0,87	1700	4,31	1600160	2,87	1317	6,69	121641
	10	1,5	23260	28,84	3200160	23,61	389842	30,98	362186
	20	17,5	655179	106,15	1600160	91,92	1282865	116,16	1201664

The table 1 shows the average results over series of 5 trials for four methods: GA, GOD, SA/SOS and SA. The values collected in the adequate columns denote: *fun* – test function, *n* – problem dimension, *f_min* – average value of the performance function calculated for 5 runs of the algorithm, *f_eval* – number of performance function evaluation needed to find the solution with the assumed stop criterion.

The hybrid method GOD proved to be very fast and robust in testing problems. It is better both in terms of speed of convergence to the solution and calculated solution's accuracy than

GA. We can say that proposed modification makes GA far more effective even for smaller size of an initial population. Results obtained for simulated annealing methods clearly show that the hybrid method SA/SOS achieves better results than simple SA.

5.2 Parallel Methods

Parallel versions of GA and SA were tested for various numbers of threads. Final quality of result and computational effort measured by number of function evaluations were compared. The goal of the tests was to calculate the solution with high level of accuracy. In sequential version of population set based methods like GA the calculations were performed for the initial population consisting of P points (P was different for different algorithms). In case of parallel versions executed on m processors, m instances of the considered algorithm were executed, each on a separate processor. Each processor transformed the population of P points. The stop criterion was the assumed number of iterations (10 000 for GA and 10 000$*n$ for SA). The results of numerical experiments are presented in table 2 and figure 2. The values collected in the table denote: n – problem dimension, nt – number of executed threads, *f_min* – average value of the performance over a serie of 5 trials, *f_best* and *f_worst*, respectively the best and worst solution.

In general, the results of calculations show that usage of parallel methods results in better accuracy, achieved in similar number of function evaluations. The improvement of final results for multiple threads was observed. In most cases the accuracy of the solution increased in 10-20%. However, parallel genetic algorithm found results even 50% better than its sequential version. Parallel simulated annealing method with synchronous communication scheme, which turned out to be the most effective version of SA-type algorithms implemented in GOOL-PV, achieved improvement of up to 65% for selected test functions.

Table 2. Comparison of sequential and parallel versions of GA and SA methods.

method		Genetic Algorithm						Simulated Annealing					
function		GR			RA			GR			RA		
n	*nt*	*f_best*	*f_min*	*f_worst*	*f_best*	*f_min*	*f_worst*	*f_best*	*f_min*	*f_worst*	*f_best*	*f_min*	*f_worst*
5	1	1,5	1,64	1,87	3,76	4,31	4,8	3,31	4,66	5,89	13,52	17,95	21,91
	2	0,95	1,55	1,93	0,58	0,91	1,08	2,81	3,75	4,46	16,06	17,78	21,50
	3	0,94	1,25	1,44	0,31	0,71	1,38	2,24	3,44	5,27	14,21	17,38	24,12
	4	0,74	1,19	1,48	0,33	0,51	0,74	2,22	3,40	4,38	8,36	13,96	18,41
10	1	5,29	5,98	6,24	26,99	28,84	31,33	22,32	27,18	30,96	66,51	71,93	73,70
	2	5,39	5,71	6,29	10,74	14,47	16,46	18,21	24,19	26,84	58,73	68,55	79,20
	3	4,98	5,62	6,23	11,04	12,55	13,71	15,53	23,05	33,51	51,8	65,43	74,73
	4	4,37	5,18	5,78	10,59	12,54	16,71	17,16	20,18	24,13	44,13	59,75	72,71
20	1	26,68	28,26	29,37	101,42	106,15	111,05	83,97	90,12	100,3	171,36	186,28	190,10
	2	20,62	24,63	29,33	77,57	83,47	92,62	63,91	81,66	90,25	165,73	178,57	193,73
	3	13,07	22,41	30,32	67,49	72,71	77,2	76,71	79,48	87,78	175,64	178,28	191,26
	4	12,75	16,98	23,35	62,1	64,9	72,53	65,71	79,51	90,04	156,24	171,25	185,82

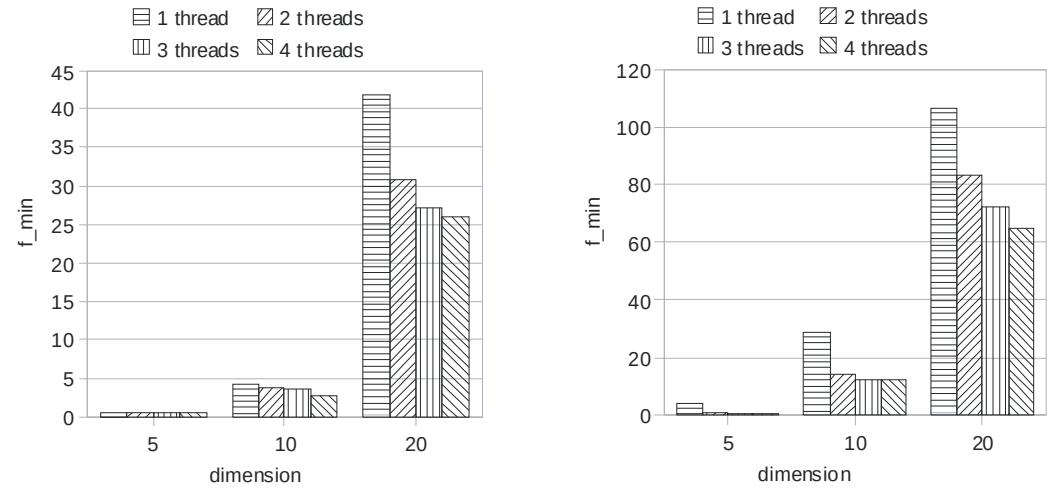

Figure 2. Solutions calculated by Genetic Algorithm a) Griewank function, b) Rastrigin function

6 Summary and Conclusions

In this paper a brief description of the software platform GOOL–PV for complex systems sequential and parallel optimization was done. We can say that presented software system can be successfully used for solving complex global optimization problems. The open design of the system architecture and extensibility to include new numerical methods make it be a useful tool for researchers and students. Our comparative study of sequential and parallel implementations of the selected global techniques shows that the parallel calculations can improve the results, and that effectiveness of a given global method strongly depends on its computer implementation and assumed attempt to its parallelization.

Appendix: test functions

Ackley function: $AC(x) = 20 + e - 20\exp\left(-\frac{1}{5}\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\right)$

The search domain: $-30 \le x_i \le 30$, the global minimum $f_{min}=0$ for $x_i = 0$, $i = 1,\ldots,n$.

Levy function:

$$LE(x) = \sin^2(\pi y_1) + \sum_{i=1}^{n-1}\left[(y_i - 1)^2(1 + 10\sin^2(\pi y_i + 1))\right] + (y_n - 1)^2(1 + 10\sin^2(2\pi y_n))$$

where $y_i = 1 + (x_i - 1)/4$. The search domain: $-10 \le x_i \le 10$, the global minimum $f_{min}=0$ for $x_i = 0$, $i = 1,\ldots,n$.

Rastrigin function: $RA(x) = 10n + \sum_{i=1}^{n} \left(x_i^2 - 10\cos(2\pi x_i) \right)$

The search domain: $-5.12 \leq x_i \leq 5.12$, the global minimum $f_{min}=0$ for $x_i = 0$, $i = 1,\ldots,n$.

Griewank function: $GR(x) = \frac{1}{40} \sum_{i=1}^{n} (x_i^2) + 1 - \prod_{i=1}^{n} \cos\left(\frac{x_i}{i} \right)$

The search domain: $-40 \leq x_i \leq 40$, the global minimum $f_{min}=0$ for $x_i = 0$, $i = 1,\ldots,n$.

Bibliography

[1] M.M. Ali, and C. Storey, "Modified controlled random search algorithms", in *International Journal of Computer Mathematics*, vol. 54, pp. 229-235, 1995.

[2] J. Arabas,, Lecture Notes In Evolution Computation, WNT, Warsaw 2001.

[3] COCONUT, project http://www.mat.univie.ac.at/~neum/glopt/coconut.

[4] A. Dekkers and E. Aarts, "Global Optimization and Simulated Annealing", in Mathematical Programming, vol. 50, pp. 367-393, 1991.

[5] Global Optimization Software: http://plato.la.asu.edu/gom.html.

[6] D.E. Goldberg, Genetic algorithms in search, optimization and machine learning, Addison–Wesley Pub. Co., 1989.

[7] A., O. Griewank,, "Generalized Descent for Global Optimization", *Journal of Optimization Theory and Applications*, Vol. 34, No.,2, pp. 11-39, 1981.

[8] R. Horst and P.M. Pardalos, *Handbook of global optimization*, Kluwer, 1995.

[9] Z. Michalewicz, "Genetic algorithms + data structures = evolution programs", Springer-Verlag, 1997.

[10] Z. Michalewicz and D.B. Fogel, *How to solve it: Modern heuristcs*, Springer-Verlag, 2000.

[11] A. Neumaier, "Complete Search in Continuous Global Optimization and Constraint Satisfaction", *Acta Numerica*, Cambridge University Press, pp. 271—369, 2004.

[12] E. Onbasoglu and L. Ozdamar, "Parallel simulated annealing algorithms in global optimization", in *Journal of Global Optimization*, vol. 19, pp. 27-50, 2001.

[13] OpenMP home page http://www.openmp.org/

[14] W.H. Press, S.A. Tukolsky, W.T. Vetterling, and B.P. Flannery, *Numerical recipes in C. The art of scientific computing*. Cambridge University Press. 1992.

[15] M. Publicewicz, E. Niewiadomska-Szynkiewicz, "GOOL – Global Optimization Object-oriented Library", proc. of KAEiOG'2003, pp. 173-181, 2003.

[16] J.W. Rogers, R.A. Donnelly, "A Search Technique for Global Optimization in Chaotic Environment", *JOTA*, vol. 61, No. 1, April 1989.

[17] R. Schaefer, *Foundations of global genetic optimization*, Springer-Verlag, Berlin, Heidelberg, 2007.

[18] Solver Technology - Global Optimization, http://www.solver.com/technology5.htm

[19] S. Tschoke, T. Polzer, "Portable Parallel Branch-and-Bound Library: PPBB-Lib, User Manual, Library Version 2.0", University of Paderborn, Germany, 1999.

[20] Yang and I. Douglas, "Simple genetic algorithm with local tuning: efficient global optimizing technique", in *Journal of Optimization Theory and Applications*, vol. 98, pp. 449-465, 1998.