OnePC: A portable software for parallel genetic algorithms used in optimization problems

Ioannis G. Tsoulos,
Department of Computer Engineering, School of Applied Technology
Technological educational institute of Epirus, Greece

Abstract—OnePC is a software entirely written in ANSI C++ using the QT library to be portable. The software implements parallel genetic algorithms that are based to a novel algorithm to tackle the global optimization problem. The proposed algorithm contains an enhanced stopping rule and a periodical application of a local search procedure. Also, the software contains a script language to assist the programmers. The article introduces the software and the underlying algorithm as well as some experimental results.

Index Terms—: Genetic algorithms, Global optimization, Parallel Computing, Programming tool

I. INTRODUCTION

The common problem of estimating the global minimum of a multi-dimensional function is given as

\[ x = \arg \min_{x \in S} f(x) \]

where \( S \) is a subset of \( R^n \) and is defined by:

\[ S = [a_1, b_1] \times [a_2, b_2] \times \cdots [a_n, b_n] \]

This problem appears in many scientific areas such as [1]-[2], chemistry [3]-[4], economics [5] etc. One common technique to tackle this is the naturally inspired method of genetic algorithms [6]. Genetic algorithms have been used in many fields [7]-[9] and they have many advantages such as:

- Adaptation in every problem
- Requirement for the objective function only and not for gradient functions
- They can be parallelized easily

The most frequent model for parallel genetic algorithms is the server - client model, such as the so called Island model [10]-[11], where clients run a local genetic algorithm and the server collects information from the clients and occasionally distributes information to them, such as the best discovered minimum. The proposed software named OnePC implements a client - server model for parallel genetic algorithms with some advanced features such as:

1. A modified stopping rule. This termination rule has been proposed also in a recent work [12] and it is general enough to be adapted in any genetic algorithm.
2. A periodical application of a local search procedure.
3. Exchange of best chromosomes between clients.

The proposed software has been written entirely in ANSI C++ using the Qt library freely available from http://qt.io in order to be portable in any operating system.

The rest of this article has as follows: in section II a detailed description of the method is given, in section III full documentation of the software is provided, in section IV some experiments that outline the usability of the software are listed and finally in section V some conclusions are presented accompanied with some guidelines for future research.

II. METHOD DESCRIPTION

A. Server algorithm

The machine denoted as server is responsible to gather from client machines the corresponding best discovered values. The algorithm executed on server is outlined below.

Initialization Step
1. Set \( g_m = \infty \)
2. Set \( N \), the number of clients

Check Step
1. If all clients have finished then
   a. Report \( g_m \) as the global minimum
   b. Terminate
2. Endif

Loop Step
1. For \( i = 1 \ldots N \) Do
   a. Obtain the minimum \( g_i \) from the client \( i \)
   b. If \( g_i < g_m \) then \( g_m = g_i \)
2. EndFor
3. GotoCheck Step
B. Client algorithm

On each client a genetic algorithm with the termination rule described in [12] accompanied with an additional local search operator is applied. The steps of the algorithm are given below:

1. Initialization Step
   a. Set \( \text{iter}=0 \), where \( \text{iter} \) is the current number of generations.
   b. Set \( N_c \), the number of chromosomes.
   c. Initialize chromosomes \( X_i, i = 1, \ldots, N_c \).
   d. Set \( IMAX \) as the maximum number of allowed generations.
   e. Set \( p_s \) as the selection rate and \( p_m \) as the mutation rate. Both rates are in range [0,1].
   f. Set \( f_i = \infty \) as the best discovered fitness.
   g. Set \( L_i \) the number of generations that should pass before the local search procedure is applied.
   h. Set \( L_C \) the number of chromosomes that will participate in local search procedure.

2. Termination Check. At every generation the variance \( \sigma^{(\text{iter})} \) of \( f_i \) is calculated. If there was no improvement of the genetic algorithm for a number of generations, then the algorithm should terminate. The stopping rule has as follows:
   \[
   \sigma^{(\text{iter})} \leq \frac{\sigma^{(\text{last})}}{2} \text{ OR } \text{iter} > IMAX \quad (3)
   \]
   Where last denotes the generation number where \( f_i \) was produced initially. If the above equation is true then goto Step 10.

3. Fitness calculation. Calculate the fitness \( f_i \) of every chromosome of the population.

4. Genetic operators.
   a. Selection procedure. The chromosomes are sorted in descending order according to their fitness value. The first \((1 - p_s) \times N_c \) chromosomes are transferred to the next generation. The rest of the chromosomes are substituted by offsprings created through crossover procedure: For every offspring two chromosomes (parents) are selected from the old population using tournament selection. The procedure of tournament selection has as follows: A set of \( N+1 \) randomly selected chromosomes is produced and the chromosome with the best fitness value in this set is selected and the others are discarded. Having selected the parents, the offsprings \( \bar{x} \) and \( \bar{y} \) are produced according to the equations
   \[
   \bar{x}_i = a_i x_i + (1-a_i)x_i \quad \text{and} \quad \bar{y}_i = a_i y_i + (1-a_i)y_i
   \]
   where \( a_i \) are random number in the range [0,1] as suggested in [13].
   b. Mutation procedure. Mutate the offsprings produced during crossover with probability \( p_m \). Suppose that the element \( i \) of a given chromosome \( x \) is denoted as \( x_i \). The new element \( x'_i \) is calculated with an equation borrowed from the popular PSO optimization method [14]:
   \[
   x'_i = c_1 r_1 (x^*_i - x_i) + c_2 r_2 (x'_i - x_i)
   \]
   where \( c_1 \) and \( c_2 \) are two positive constants (acceleration coefficients), \( r_1 \) and \( r_2 \) are random numbers in the range [0,1], the vector \( x^b \) is a copy of the best so far position of chromosome \( x \) and \( x^l \) is the best discovered chromosome so far.

   c. Replace the \( p_s \times N_c \) worst chromosomes in the population with the offsprings created by the genetic operators.

5. Set \( \text{iter} = \text{iter} + 1 \).

6. Local Search Step
   a. If \( \text{iters} \mod L_i = 0 \) then
      i. Select randomly \( L_C \) chromosomes from the genetic population and create set \( L_z \) from these chromosomes.
      ii. For every chromosome \( X_i \in L_z \)
         1. Select randomly another chromosome \( Y \) from the population
         2. Create an offspring of \( X_i \) and \( Y \). Denote this chromosome as \( Z \)
         3. Obtain the fitness \( f(z) \) of the chromosome \( Z \)
         4. If \( f(z) < f(x_i) \) then \( X_i = Z \)

   b. EndIf

7. Obtain the best value in the population, denoted as \( f_i \) for the corresponding chromosome \( x_i \).
8. Send \( (x_i, f_i) \) to the server machine.
10. Send \( (x_i, f_i) \) to the server machine.
11. Terminate.

III. SOFTWARE DOCUMENTATION

In this section a detailed description of the main parts of the software is provided. More detailed information about the software that includes some screenshots can be found in the URL http://itsoulos.itiep.gr/OnePCSite/index.html. Also a last copy of the software is provided in https://github.com/itsoulos/OnePC

A. User provided functions

The user should code the objective function in C++. The C++ files should have the following command before any function in the file

\[
\text{extern "C" }
\]

and the line

\[
}\}
\]

after them. The user should supply the following functions:

1. \text{getDimension()}. It is an integer function which returns the dimension of the objective function.
2. \text{getLeftMargin()}. It is an integer function which returns the dimension of the objective function.
3. \texttt{getrightmargin(right)}. It is an integer function which returns the dimension of the objective function.

4. \texttt{funmin(x)}. It is an integer function which returns the dimension of the objective function.

As an example consider the Rastrigin function:

\[ f(x_1, x_2) = x_1^2 + x_2^2 - \cos(18x_1) - \cos(18x_2) \]

The code for this objective function is shown in Fig. 1. Also the user may optionally provide the following functions:

1. \texttt{void \textit{init(QJsonObject params)}}. This function initializes the objective problem and it is not mandatory. In params parameter user stores some useful parameters for the problem. The user should use the include directive \#include \textless QJsonObject \textgreater at the beginning of the file. As an example consider the initialization function for the problem of energy potential shown in Fig. 2, where some critical parameters of the objective function are initialized through the \textit{init()} function.

2. \texttt{double done(double \*x)}. This function should be called at the end of optimization. The double parameter \textit{x} is the vector of global minimum discovered by the optimization process. The function should return the final value of the objective function. It is useful when a local optimization algorithm is required at the end of the optimization of in the neural network case, where the neural network should be applied to the test set.

The objective problem is written in ANSI C++ and it should build as a shared library. The use can use any programming tool to achieve that, but the best choice is to use Qt build environment for that. The project file is used in Qt programming environment to build the associated project. In our case is used to build the optimization function. An example for building the potential optimization function is illustrated in Fig 3.

\section*{B. The program OnePCServer}

\section*{Installation}

The user should issue the following commands to build the program OnePC Client (under some Unix machine):

1. Download the code (OnePcClient.tar.gz)
2. gunzip OnePcClient.tar.gz
3. tar xvf OnePcClient.tar
4. cd OnePcClient
5. qmake
6. make

\section*{User Interface}

The graphical interface of the program is written entirely in Qt and it contains for main tabs:

1. \textbf{Info tab}. The tab contains information about the machine running the server. The most important information from this tab is the IP of the server and the port where it is running. This information should be used by the clients that want to connect to genetic server. Also, the user can run OneScript programs using the button LOAD SCRIPT.

2. \textbf{Clients tab}. This is the tab where the user can start the parallel genetic algorithm by pressing the button Run Experiment. This button sends on every client the objective function. The function will be sent in the corresponding format of every client depending of the running operating system. Afterwards each client starts the genetic algorithm. The Clients tab displays also the following information for every client:
   a. Name of the client
   b. Operating system of the client
   c. Status of the client (running, waiting, terminated, paused)
   d. Discovered global minimum from the client

3. \textbf{Problems tab}. The information about the stored objective problems is outlined in this tab. This information is stored in a sqlite3 database named gaserver.db3 in the same folder with the running server. The user can load compiled objective problems in shared library format from the hard disk and he can also change or add parameters to the objective problem through the relevant dialog.

4. \textbf{Messages tab}. This tab displays debug information for every client as well as about the status of the running server.

\section*{C. The program OnePCClient}

\section*{Installation}

The user should issue the following commands to build the program OnePC Client (under some Unix machine):

1. Download the code (OnePcClient.tar.gz)
2. gunzip OnePcClient.tar.gz
3. tar xvf OnePcClient.tar
4. cd OnePcClient
5. qmake
6. make

\section*{User Interface}

The graphical application is written entirely in Qt in order to be portable. Nevertheless, the user can start the client in command line mode by starting the executable with the command line \textit{--gui=no}. The graphical application is organized in three tabs:

1. \textbf{Connection tab}. In this tab the user should supply the ip of the server in the Host textfield as well as the running port. Without this information the client cannot connect to the server. Also, the user can provide a name for the client in order to distinguish them from other running clients.

2. \textbf{Run tab}. Under Run tab the user can modify some of the most critical parameters of the genetic algorithm such as chromosomes, generations etc. Also, the user can monitor the progress of the genetic algorithm during the execution of the algorithm.

3. \textbf{Messages tab}. Some debug information is displayed in this tab.

\section*{Command line options}

The user can use command line arguments for the application.

1. \textit{--generations=g}. The integer value \textit{g} specifies the maximum number of generations allowed for the genetic algorithm. The default value is 200.

2. \textit{--chromosomes=c}. The integer value \textit{c} specifies the amount of chromosomes used in the genetic
population. The default value for this parameter is 200.

3. \texttt{selectionrate}=s. The double parameter s determines the selection rate for the genetic algorithm. The default value is 0.90 (90%), which means that 90% of the population will be copied intact to the next generation.

4. \texttt{mutationrate}=m. The double parameter m determines the mutation rate used in genetic algorithm. The default value is 0.05 (5%).

5. \texttt{serverip}=s. The string parameter s specifies the ip server of the OnePC server.

6. \texttt{serverport}=p. The integer parameter p specifies the port of the OnePC server.

7. \texttt{gui}=b. The boolean value b can specifies if gui will be used for the client. If b is yes or true the gui will be used otherwise it will not.

8. \texttt{seed}=s. The integer parameter s specifies the seed for the random generator.

9. \texttt{machinename}=name. The string parameter name specifies the name of the client that will be displayed under tab Clients in OnePC server.

10. \texttt{parallelchromosomes}=c. The integer parameter c specifies the number of chromosomes that will be exchanged between this client and the other clients in the parallel population. The default value for this parameter is 10.

11. \texttt{localsearchchromosomes}=c. The integer parameter c specifies the amount of chromosomes that will take part into local search step of the genetic algorithm. The default value for this parameter is 20.

12. \texttt{localsearchgenerations}=g. The integer parameter g specifies the amount of generations that will be executed before the local search step of the genetic algorithm. The default value for this parameter is 50.

D. The language OneScript

The language OneScript can be used to simplify the execution of a series of problems in OnePC without the interference of the programmer. The commands can be written using capital or lowercase letters. For the time being the language has a simple set of commands which are:

1. \texttt{SET PROBLEM name}. Sets the current problem to name.

2. \texttt{SET PARAMETER paramName paramValue}. Change the value of the parameter paramName to paramValue.

3. \texttt{RUN}. Executes the current problem using the attached clients.

4. \texttt{PRINT FILE}. Print the current global minimum to file FILE.

As an example of a script in OneScript consider the program listed in Fig. 4 used to set the objective problem to SINU.

```cpp
#include <math.h>
extern "C"
{
    int getdimension()
    {
        return 2;
    }

    void getleftmargin(double *x)
    {
        x[0]=-1;
        x[1]=-1;
    }

    void getrightmargin(double *x)
    {
        x[0]=1;
        x[1]=1;
    }

do double funmin(double *x)
    {
        return x[0]*x[0]+x[1]*x[1]-cos(18.0*x[0]) -cos(18.0*x[1]);
    }
}
```

\textit{Figure 1. The Rastrigin function.}

double potentialepsilon=1.0;
double potentialsigma=1.0;
inntatoms=10;
inttolminiters=2001;
QStringoutputfile="potential.txt";
double *xx=NULL;
double *yy=NULL;
double *zz=NULL;
void init(QJsonObjectobj)
{
    if(obj.contains("natoms"))
        natoms=obj["natoms"].toString().toInt();
    if(obj.contains("tolminiters"))
        tolminiters=obj["tolminiters"].toString().toInt();
    if(obj.contains("outputfile"))
        outputfile=obj["outputfile"].toString();
    xx=new double[natoms];
    yy =new double[natoms];
    zz =new double[natoms];
}
```

\textit{Figure 2. Initialization function of Potential problem.}
IV. EXPERIMENTS

The method was tested on some test functions from the relevant literature as well as on the Lennard Jones potential problem. The results are compared against the well-known software for parallel computation called GALib-mpi[18]. The experiments were performed 30 times using different seed for the random generator each time and averages were taken. The number of chromosomes was set to 200 and the maximum number of allowed generations was set to 2000.

A. Experiments on Test Function

The following test functions were used in the experiments:

1. **Giks.** \( f(x) = \text{Giks}(x, n, w) \) is a function with \( w \) local minima described in [19]. \( x \in [-1,1]^n, n \in [2,100] \). In the conducted experiments the cases of \( n = 2,3,4 \) with 50 local minima were used.

2. **Test2n.** The function is given by
   \[
   f(x) = \frac{1}{2} \sum_{i=1}^{n} (x_i^4 - 16x_i^2 + 5x_i) \quad \text{with} \quad x \in [-5,5]^n.
   \]
   The function has \( 2^n \) local minima in the specified range. In the conducted experiments the cases of \( n = 5,6,7 \) was considered.

3. **Sinusoidal.** The function \( f(x) \) is given by
   \[
   f(x) = -\left( -2.5 \prod_{i=1}^{n} \sin(x_i - z) + \prod_{i=1}^{n} \sin(5(x_i - z)) \right)
   \]
   with \( 0 \leq x_i \leq \pi \) and \( z = \frac{\pi}{6} \). The global minimum is -3.5.

4. **The Chemical Equilibrium.** The problem is described in [15] and it is described by the following set of equations:
   \[
   \begin{align*}
   x_1 + x_2 + x_3 &= 3x_1 \\
   2x_1x_2 + x_1 + x_2 + x_3 + R_{0x_2} - R_{x_2} + 2R_{0x_2}x_1 + R_{x_1x_2}x_2 + R_{x_2x_4} & \\
   2x_2x_3 + 2R_{0x_2}^2 - 8x_2 + R_{x_2} + R_{x_2}x_3 + R_{x_3}x_2 & \\
   R_{0x_4} + 2x_3 + 4R_{x_3} & \\
   \end{align*}
   \]
   \[
   x_1(x_2 + 1) + R_{10x_2} + x_2 + x_3 + x_4 + R_{x_2} + R_{x_3} + x_4 + 1 + R_{x_3} + R_{x_1x_2} + R_{x_2x_4}
   \]
   where

The corresponding objective function is the summation of the absolute values of all equations in the system i.e.

\[
 f(x) = \sum_{i=1}^{n} |f_i(x)|
\]

The global minimum is 0.

5. **The Kinematic Application.** This function is described in [15] for the inverse position problem for a six – revolute – joint problem in mechanics. The problem is provided as a system of equations:

\[
\begin{align*}
 x_1^2 + x_i^2 &= 0 \\
 a_{11}x_1x_3 + a_{12}x_1x_4 + a_{13}x_2x_3 + a_{14}x_2x_4 + a_{15}x_2x_3 &= 0 \\
 a_{21}x_2x_3 + a_{22}x_2x_4 + a_{23}x_1x_3 + a_{24}x_1x_4 + a_{25}x_1x_3 + a_{26}x_1x_4 &= 0 \\
 a_{31}x_3x_4 + a_{32}x_3x_4 + a_{33}x_1x_3 + a_{34}x_1x_4 + a_{35}x_1x_3 + a_{36}x_1x_4 &= 0 \\
 a_{61}x_1x_3 + a_{62}x_1x_4 + a_{63}x_2x_3 + a_{64}x_2x_4 + a_{65}x_2x_3 + a_{66}x_2x_4 &= 0 \\
 a_{81}x_1x_3 + a_{82}x_1x_4 + a_{83}x_2x_3 + a_{84}x_2x_4 + a_{85}x_2x_3 + a_{86}x_2x_4 &= 0 \\
 \end{align*}
\]

Where 1 ≤ i ≤ 4 and the table \( \delta_{ij} \) is

\[
\begin{array}{cccc}
0.249150 & 0.125016 & -0.635550 & 1.489977 \\
1.609153 & -0.68660736 & -0.1157199 & 0.2360623 \\
0.27942343 & -0.11922812 & -0.6664043 & 1.3281073 \\
1.3438016 & -0.71994047 & 0.110362 & -0.2586458 \\
0.0 & -0.43241927 & 0.2970702 & 1.165172 \\
0.40802694 & 0.0 & 1.2587767 & -0.26908 \\
-0.80052768 & 0.0 & -0.629388 & 0.53816 \\
0.0 & -0.86483555 & 0.581404 & 0.582858 \\
0.074045 & -0.03715727 & 0.195946 & -0.208169 \\
-0.083050 & 0.0354368 & -1.2280034 & 2.688632 \\
-0.38615961 & 0.085303482 & 0.0 & -0.699103 \\
-0.75526603 & 0.0 & -0.079034 & 0.3574441 \\
0.50421068 & -0.039251967 & 0.026387 & 1.2499911 \\
-1.0916287 & 0.0 & -0.057131 & 1.467736 \\
0.0 & -0.43241927 & 1.1628081 & 1.165172 \\
0.04920729 & 0.0 & 1.2587767 & 1.0763397 \\
0.04920729 & 0.01397301 & 2.162575 & -0.6968688 \\
\end{array}
\]

The corresponding objective function is the summation of the absolute values of all equations in the system i.e.

\[
 f(x) = \sum_{i=1}^{n} |f_i(x)|
\]

The global minimum is 0.
Again, the corresponding objective function is the summation of the absolute values of all equations in the system i.e.

\[ f(x) = \sum_{i=1}^{n} |f_i(x)| \]

The global minimum is 0.

The results from the applications of OnePC and GaLib to the problems above are presented in Table I for two processors, in Table II for four processors and in Table III for eight processors. The cells denote average number of generations and the figures in parentheses denote the fraction of runs that located the global minimum and were not trapped in one of the local minima. Absence of this fraction denotes 100% success in locating the global minimum. In all tables the column FUNCTION denotes the function name, the column GaLib denotes the results from the application of GALIB and the column ONEPC denotes the results from the application of the proposed software.

As we can deduce from the experimental results the proposed software seems to require lower number of generations than GaLib to discover the global minimum. Also the fraction of runs that discovered the global minimum is higher than GaLib.

B. The case of Lennard Jones Potential

The molecular conformation corresponding to the global minimum of the energy of N atoms interacting via the Lennard-Jones potential [16] is used as a test case here. The function to be minimized is given by:

\[ V_{ij}(r) = 4\varepsilon \left[ \left( \frac{\sigma}{r} \right)^{12} - \left( \frac{\sigma}{r} \right)^{6} \right] \]

The function is minimized firstly with OnePC and afterwards the local search procedure Tolmin (a BFGS variant of Powell [17]) is used to enhance the detected minimum. Success rates of discovering the global minimum by GaLib are presented in Table IV and success rates obtained by the proposed method are presented in Table V. Again in most cases the fraction of runs that discovered the global minimum is higher than GaLib.

V. CONCLUSIONS

A portable software for global optimization was introduced with the following features:

1. It can be installed in most operating systems.
2. There is no need for parallel libraries such as OpenMPI[20].
3. The system continues to work even if some of the nodes have lost connection.
4. The clients can operate with or without GUI.
5. The user can control the initialization parameters of the objective problems through the init() procedure.
6. The server can use a script language to control the optimization procedure.

Future research may include:

1. Additional commands for the script language of the server.

### Table I: Experimental results using two processors for a series of optimization problems

<table>
<thead>
<tr>
<th>PROBLEM</th>
<th>GaLib</th>
<th>OnePC</th>
</tr>
</thead>
<tbody>
<tr>
<td>GKLS350</td>
<td>310.50</td>
<td>205.18</td>
</tr>
<tr>
<td>GKLS450</td>
<td>370.20(0.30)</td>
<td>393.78(0.77)</td>
</tr>
<tr>
<td>GKLS550</td>
<td>1548.70(0.93)</td>
<td>569.80(0.80)</td>
</tr>
<tr>
<td>SINU16</td>
<td>1933.50</td>
<td>1633.63</td>
</tr>
<tr>
<td>SINU32</td>
<td>1933.57(0.90)</td>
<td>1644.18</td>
</tr>
<tr>
<td>SINU64</td>
<td>272.50(0.17)</td>
<td>155.68(0.17)</td>
</tr>
<tr>
<td>TEST2N4</td>
<td>543.60</td>
<td>425.02</td>
</tr>
<tr>
<td>TEST2N5</td>
<td>857.63</td>
<td>521.22</td>
</tr>
<tr>
<td>TEST2N6</td>
<td>1037.37</td>
<td>786.27</td>
</tr>
<tr>
<td>TEST2N7</td>
<td>1207.90(0.93)</td>
<td>891.7</td>
</tr>
<tr>
<td>CHEMICAL</td>
<td>1917.25(0.93)</td>
<td>1415.31</td>
</tr>
<tr>
<td>KINEMATIC</td>
<td>1976.37</td>
<td>1601.81</td>
</tr>
<tr>
<td>COMBUSTION</td>
<td>1865.67</td>
<td>1409.19</td>
</tr>
</tbody>
</table>

### Table II: Experimental results using four processors for a series of optimization problems

<table>
<thead>
<tr>
<th>PROBLEM</th>
<th>GaLib</th>
<th>OnePC</th>
</tr>
</thead>
<tbody>
<tr>
<td>GKLS350</td>
<td>429.93</td>
<td>255.58</td>
</tr>
<tr>
<td>GKLS450</td>
<td>424.87(0.33)</td>
<td>359.73(0.90)</td>
</tr>
<tr>
<td>GKLS550</td>
<td>1322.43(0.90)</td>
<td>585.87</td>
</tr>
<tr>
<td>SINU16</td>
<td>1867.20(0.90)</td>
<td>1611.76</td>
</tr>
<tr>
<td>SINU32</td>
<td>1740.63(0.90)</td>
<td>1657.10</td>
</tr>
<tr>
<td>SINU64</td>
<td>339.50(0.23)</td>
<td>72.29(0.30)</td>
</tr>
<tr>
<td>TEST2N4</td>
<td>712.67</td>
<td>342.78</td>
</tr>
<tr>
<td>TEST2N5</td>
<td>984.57(0.97)</td>
<td>498.70</td>
</tr>
<tr>
<td>TEST2N6</td>
<td>983.47(0.90)</td>
<td>624.65</td>
</tr>
<tr>
<td>TEST2N7</td>
<td>1015.67(0.70)</td>
<td>957.80</td>
</tr>
<tr>
<td>CHEMICAL</td>
<td>1825.37</td>
<td>1319.69</td>
</tr>
<tr>
<td>KINEMATIC</td>
<td>1903.94</td>
<td>1644.89</td>
</tr>
<tr>
<td>COMBUSTION</td>
<td>1836.54</td>
<td>1241.43</td>
</tr>
</tbody>
</table>

### Table III: Experimental results using eight processors for a series of optimization problems

<table>
<thead>
<tr>
<th>PROBLEM</th>
<th>GaLib</th>
<th>OnePC</th>
</tr>
</thead>
<tbody>
<tr>
<td>GKLS350</td>
<td>418.17</td>
<td>224.49</td>
</tr>
<tr>
<td>GKLS450</td>
<td>626.40(0.40)</td>
<td>373.13</td>
</tr>
<tr>
<td>GKLS550</td>
<td>1291.00(0.83)</td>
<td>592.63</td>
</tr>
<tr>
<td>SINU16</td>
<td>1933.26(0.93)</td>
<td>1642.59</td>
</tr>
<tr>
<td>SINU32</td>
<td>1933.63(0.87)</td>
<td>1678.78</td>
</tr>
<tr>
<td>SINU64</td>
<td>373.87(0.23)</td>
<td>55.38(0.67)</td>
</tr>
<tr>
<td>TEST2N4</td>
<td>461.77</td>
<td>375.70</td>
</tr>
<tr>
<td>TEST2N5</td>
<td>947.70(0.97)</td>
<td>506.12</td>
</tr>
<tr>
<td>TEST2N6</td>
<td>1092.13(0.93)</td>
<td>629.29</td>
</tr>
<tr>
<td>TEST2N7</td>
<td>1097.43(0.73)</td>
<td>883.76</td>
</tr>
<tr>
<td>CHEMICAL</td>
<td>1908.33</td>
<td>1383.93</td>
</tr>
</tbody>
</table>
Table IV Success rate results for the Potential problem with GaLib

<table>
<thead>
<tr>
<th>NATOMS</th>
<th>C2</th>
<th>C4</th>
<th>C8</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>6</td>
<td>27%</td>
<td>37%</td>
<td>50%</td>
</tr>
<tr>
<td>7</td>
<td>70%</td>
<td>73%</td>
<td>97%</td>
</tr>
<tr>
<td>8</td>
<td>93%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>9</td>
<td>63%</td>
<td>87%</td>
<td>93%</td>
</tr>
<tr>
<td>10</td>
<td>53%</td>
<td>73%</td>
<td>97%</td>
</tr>
<tr>
<td>12</td>
<td>63%</td>
<td>83%</td>
<td>93%</td>
</tr>
</tbody>
</table>

Table V Success rate results for the Potential problem with OnePC

<table>
<thead>
<tr>
<th>NATOMS</th>
<th>C2</th>
<th>C4</th>
<th>C8</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>6</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>7</td>
<td>80%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>8</td>
<td>60%</td>
<td>80%</td>
<td>100%</td>
</tr>
<tr>
<td>9</td>
<td>70%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>10</td>
<td>40%</td>
<td>80%</td>
<td>90%</td>
</tr>
<tr>
<td>12</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
</tbody>
</table>

REFERENCES