A Combination of Parallel Computing and Object-Oriented Programming to Improve Optimizer Robustness and Efficiency

Guido Buzzi-Ferraris, Flavio Manenti

Dipartimento di Chimica, Materiali e Ingegneria Chimica “Giulio Natta”
Poltectnico di Milano, Piazza Leonardo da Vinci 32, 20133 Milano, ITALY
E-mail: guido.buzziferraris@polimi.it

Abstract

This research activity is mainly aimed at showing potentialities in coupling object-oriented programming with parallel computing. Wide margins of benefits could be obtained in algorithm efficiency and robustness with a relative small programming effort. The case of unconstrained multi-dimensional optimization is proposed as quantitative example.

Keywords: Robust optimization, Multimodality, Narrow valleys, Discontinuous functions, Parallel Computing.

1. Introduction

We are undergoing two silent revolutions that directly involve Process Systems Engineering (PSE) and Computer-Aided Process Engineering (CAPE) communities, besides many other scientific and industrial areas: the object-oriented programming and the parallel computing on personal computer. Both these transformations have been widely discussed in the literature as they significantly modify the numerical analysis as it was conceived since the second part of the previous century and the way to apply numerical methods and algorithms for solving more and more complex problems and multifaceted issues.

Nevertheless, since it is not clearly stated in the current literature, it is worth remarking that the parallel computing is easy to integrate in object-oriented programming and their combination seems particularly appealing as many objects generated by the same class might run simultaneously on different processors or cluster nodes. By thinking parallel and object-oriented both together, it is possible to write by new many algorithms which were not considered for solving numerical problems because of their reduced performances in the procedural programming and sequential computing.

Thus, this research activity specifically deals with the development of very robust optimizers that exploit:

- All features of object-oriented programming (Buzzi-Ferraris, 1994), which allow going beyond the procedural programming and its limitations.
- The shared memory nowadays commonly available on multi-processor machines (distributed memory machines are not considered for the time being, even though the same reasoning here described can be extended to this branch of parallel computing).

Motivation and practical interests in some scientific and industrial areas are briefly reported in Paragraph 2. Basic concepts of coupling parallel computing with object-
oriented programming for improving the optimizer robustness and efficiency are stated in Paragraph 3. Some literature tests involving multidimensionality, strong and weak multimodality, very narrow valleys, and functions that are undefined in some regions are proposed in Paragraph 4.

2. Motivation and Practical Interests

Looking at the increasing spread of multi-processor machines as well as the larger and larger amount of processors available on the common PCs, it is easy to see how the period we are living is very similar to the one of 1970s when the most powerful (and very large-size) machines were gradually replaced by smallest personal computers with reduced computational power, but with a large impact on research activities for their faster spread, reduced costs, reasonably good performances, and especially for their higher slope in innovation. In our opinion, but it is easy to find out some confirmations yet, shared memory machines will have a faster evolution than distributed memory architectures and looking forward this, we preferred to use openMP directives rather than MPI ones in starting exploiting parallel computing, at least for this preliminary research activity.

In any case, the result is practically the same by using one or the other set of directives as efficiency and robustness of many algorithms can be significantly improved so to increase performance solution of a series of industrial issues and to allow moving from reacting to predicting technologies applied to industrial plants (Manenti, 2009) and from those solutions still performed off-line to their on-line application (White, 2001) by preserving the robustness of the selected methods.

To quote some examples typical of process industry and PSE/CAPE communities, improvements in optimizer efficiency and robustness can provide practical benefits in data reconciliation (Arora and Biegler, 2001; Bagajewicz, 2003), in data regression (Buzzi-Ferraris and Manenti, 2009a, 2009b, 2010b; Manenti and Buzzi-Ferraris, 2009), in solving nonlinear algebraic or differential systems (Cuoci et al., 2007; Manenti et al., 2009), or in the supply chain management optimization levels (Dones et al., 2009; Lima et al., 2009; Manenti and Rovaglio, 2008).

3. Exploiting Shared Memory to Improve Efficiency and Robustness

Conventional programs easily fail when some specific families of optimization problems have to be solved. Very robust optimizers are required in these cases:

- When the function is multimodal and the global optimum is required
- The function and/or its derivatives are discontinuous
- The function cannot be approximated by a quadric in correspondence with the optimum
- Very narrow valleys (or steep walls) are present
- The function is undefined in some regions and the domain cannot be analytically described

Although no one can ensure the global optimum is found, a robust algorithm should be effective in tackling all previous situations.

Let us start analyzing the problem of very narrow valleys. From this perspective, the OPTNOV’s method (Buzzi-Ferraris, 1967) seems one of the most appealing approach. It is important to realize the reason that makes traditional methods such as Simplex (Nelder and Mead, 1965), Hooke-Jeeves (Hooke and Jeeves, 1961), Rosembrock (Rosenbrock, 1960), Quasi-Newton algorithms and so on ineffective when the function valleys are particularly narrow. For example, Rosembrock’s method is based on the
A Combination of Parallel Computing and Object-Oriented Programming to Improve Optimizer Robustness and Efficiency

rotation of axes of search so to follow the bottom of the valley. Since one of rotated axes is adopted as search direction, it may occur that moving along it does not bring to any function improvement when the valley is very narrow as shown in Figure 1.

![Figure 1. Rosembrock’s method fails with very narrow valleys.](image)

To exploit the search direction that inaccurately detects the bottom of the valley, it is necessary to change the point of view. OPTNOV’s method is based on some simple ideas that make it particularly robust and efficient in the case of very narrow valleys:

- Whatever optimization algorithm is able to find the bottom of the valley by starting from a point outside the same valley
- The line joining two points on the bottom of the valley is a reasonable valley direction; therefore a point projected along such a direction has good probabilities to be close to the valley
- Nevertheless, this valley direction must not be used as direction of one-dimensional search, rather as a direction which a new point projection must be carried out along
- This new point should not be discarded even though it is worse than the previous one, rather it is the new starting point for the search.
- This search must be performed in the sub-space orthogonal to the valley direction to prevent the problem of having small steps

This philosophy is particularly effective in an object-oriented programming coupled with parallel computing as many reduced optimizations must be carried out starting from distinct points and they can be independently solved each other. Consequently, this philosophy of simultaneously solving different optimization problems by starting from distinct guesses allows rationally facing even the global minimum paradigm.

The concept to build up a program for effectively tackling all aforementioned issues is rather trivial as it is possible to develop an optimizer consisting of $N$ objects, where $N$ is the number of available processors and each of them uses in turn an optimizer reasonably robust.

Hence, two distinct problems must be solved: the first is the selection of points used in the $N$ objects as initial guess and the second is which optimizer to use within each of these $N$ objects; it is worth remarking that even this optimizer must be opportunely robust: to manage possible first- and second-order discontinuities of the function; to overcome possible regions where the same function is undefined; to ensure the global minimum in one-dimensional searches is found; and to efficiently tackle the problem of slightly narrow valleys.

For the sake of clarity, let us call inner the optimizer used within each of the $N$ objects and outer the one managing the overall optimization problem. The outer optimizer only is discussed in this paper.
The problem of searching for the global optimum of the overall problem and to overcome its possible narrow valleys are both tasks of the outer optimizer. The following strategy is proposed to manage the \( N \) objects: three objects are required for applying OPTNOV’s philosophy whereas the remaining \( N - 3 \) objects are selected by using the same techniques employed in optimal experimental design (Buzzi-Ferraris, 1999; Buzzi-Ferraris and Manenti, 2009a, 2010a, 2010b; Manenti and Buzzi-Ferraris, 2009). Therefore, there is the lower bound of using four processors (QUAD CORE machines). The outer optimizer collects initial and arrival points of each inner object and it selects the two points having the best performances among all those ones collected. If these two points are significantly close, the best third, fourth… is selected in spite of the second to avoid any ill-conditioning while detecting the valley direction.

![Figure 2. Points A and B are on the bottom of the valley (A is the best one); points I, II, and III are the possible point projections along the valley direction.](image)

Distances \( \delta \) and \( \epsilon \) among points can be reduced or expanded according to the results: for example, if the point III brings to a better inner optimum, distances are expanded. Points from the fourth to the \( N-th \) are selected so to have the farthest points against all the collected ones. This selection is efficiently carried out by using those techniques adopted and proven for the optimal design of experiments. The following procedure is adopted as stop criterion. At each iteration, the number of points in the neighborhood of the optimum (given a tolerance value) is checked. If such a number is reasonable (according to an assigned value), a possible solution is reached. Theoretically, the number of points should be in the order of magnitude of the optimization problem dimensions, but it is preferable to use smaller numbers when the optimization size is large.

4. Numerical Tests

Many numerical tests were carried out to check the algorithm robustness for problems of different dimensions. Tests of Table 1 are well-known literature functions for:

- “Strong” multimodality issues: all directions are directions of function increase in correspondence with local minima. Both Rastrigin (1) and Haupt (2) functions were adopted and reported in Figure 3.
- “Weak” multimodality issues: the function is constant along at least one direction in correspondence of some local minima. Michalewicz’s function (3) was adopted and reported in Figure 3.
- Discontinuities and regions where the function is not defined (4). Figure 4 shows the function against the variable \( x_i \) for the optimal value of \( x_i \).
- Extremely narrow valleys: valleys consisting of steep walls make the search of the minimum a problematic issue for many optimizers. Buzzi-Ferraris’s function (5) is adopted and reported in Figure 4.

\[
F_{RASTRIGIN} = -110 \cdot n + \sum_{i=1}^{n} \left( x_i^2 - 10 \cos(2\pi x_i) \right)
\]  

(1)
A Combination of Parallel Computing and Object-Oriented Programming to Improve Optimizer Robustness and Efficiency

\[ \begin{align*}
F_{\text{HAUPT}} &= -a, \quad a > 0 \\
F_{\text{HAUPT}} &= 0, \quad a \leq 0 
\end{align*} \]

where: \( a = \prod_{i=1}^{n} \left( \sqrt{x_i} \sin(2\pi x_i) \right) \) (2)

\[ F_{\text{MICHALEWICZ}} = -\sum_{i=1}^{n} \sin(x_i) \left( \sin \left( \frac{i \cdot x_i^2}{\pi} \right) \right)^{2n} \] (3)

\[ F_{\text{BUZZI-FERRARIS},A} = \sqrt{945 + x_1 \left[ 1689 + x_2 \left( -950 + x_1 \left( 230 + x_1 \left( -25 + x_1 \right) \right) \right) \right] + e^{-x_1} + 10|x_2 - 10x_1| + 10|x_1 - 6|} \] (4)

\[ F_{\text{BUZZI-FERRARIS},B} = \left[ x_2 - 10000(x_1 - 1)(x_1 - 3)(x_1 - 5)(x_1 - 7)(x_1 - 9) \right]^2 + (x_1 - 8)^2 \] (5)

Figure 3. Two-dimensional Haupt (left), Rastrigin (middle), and Michalewicz (right) functions

Figure 4. Buzzi-Ferraris’s functions (A, left; B, right) to test optimizer robustness

<table>
<thead>
<tr>
<th>Table 1. Optimization tests</th>
</tr>
</thead>
<tbody>
<tr>
<td>Starting point</td>
</tr>
<tr>
<td>-----------------</td>
</tr>
<tr>
<td>Rastrigin ( n = 2 )</td>
</tr>
<tr>
<td>Rastrigin ( n = 10 )</td>
</tr>
<tr>
<td>Haupt</td>
</tr>
<tr>
<td>Michalewicz ( n = 2 )</td>
</tr>
<tr>
<td>Michalewicz ( n = 10 )</td>
</tr>
<tr>
<td>Buzzi-Ferraris A</td>
</tr>
<tr>
<td>Buzzi-Ferraris B</td>
</tr>
</tbody>
</table>
5. Conclusions and Future Developments
This preliminary research activity shows the way and reports some benefits coming from the interaction of parallel computing and object-oriented programming. Specifically, the example of C++ class for robust optimization that could generate a series of objects so that each of them could run on a specific processor by increasing the same optimizer robustness with a small programming effort is proposed.

References