Controlled Random Search Parallel Algorithm for Global Optimization with Distributed Processes on Multivendor CPUs

Salvatore Rinaudo†, Francessco Moschella†, and Marcello A. Anile‡
† SGS-THOMSON Microelectronis, Stradale Primo Sole 50, 95121 Catania (Italy)
‡Dipartimento di Matematica, Viale Andrea Doria, 95125 Catania (Italy)

Abstract— **This work presents the implementation of the Controlled Random Search (CRS) [1], [2] algorithm for global optimization within a parallel processing environment consisting of a cluster of multivendor workstations. This allows us to extract and optimize parameters or calibrate a general simulator, without needing the analytical form of the implemented model.**

I. Introduction

In many areas of research and design simulators are a crucial tool for determining the effect of parameter variations on the output of a given system. The reliability of a simulator depends on the accuracy of the implemented models (e.g. physical or otherwise) and, in particular, the model parameters. Generally, the method used to extract parameters from an event, is based on the Least Squares Method, i.e. in minimizing the l^2 norm of the difference between the output and the measured (or required) value. Many commercially available simulation tools in the microelectronics industry are endowed with optimization programs, usually based on Least Squares Methods coupled with a numerical solver for minimizing, such as the normal equations or gradient methods. These optimization codes are strictly linked to the whole simulation package and cannot be easily adapted to the various requirements of an industrial environment. For instance, in some industrial applications, one would like to have a global optimizer, which, being computationally costly, is usually not available in the commercial package. Howewer the greater computational cost of global optimization could be tolerable in an industrial context if efficient use is made of the computing power of a given design unit (e.g. by an appropriate use of a cluster of multivendors workstations). Another example of great interest is that of optimizing a cost function which is computed by using several simulation codes which are not integrated in a single software package and have been provided by different software vendors. In this article we present a global numerical optimizator implemented in a software package, named *exemplar*, based on the CRS method [1], [2] suitably modified by us in order to obtain better performance. This method has been implemented in a parallel version by utilizing a given cluster of multivendor workstations, already used in the design center (therefore no further HW and SW investment has been necessary).

II. CRS Method

Given a function of n variables (in our case, the residual function $R(\vec{P})$), an initial search domain V, is defined by specifying limits to each variable. A predetermined number N of trial points are chosen at random over V. The function is evaluated at each trial and the position and corresponding value stored in an array A. After building the initial array A, at each iteration a new trial point is selected, the function is evaluated in $\vec{P}$ (f_P) and if its value is lower than the maximum one (f_M) stored in A, this last one is replaced by $\vec{P}$ and its function value by f_P. As the algorithm proceeds, the current set of N stored points tend to cluster around a minima. The CRS procedure is very simple, as it does not need either an initial guess or knowledge of derivatives.

III. CRS on a parallel processing environment

Since this procedure chooses the next trial point starting from a random search, each iteration does not depend on the previous one. So if we have M available CPU we can choose M trial points and evaluate their function values concurrently. The algorithm used for parallel processing could be described in the following steps:

- **Step 1**
  Choose **M** points at random over **V**, with M the number of available CPU, and concurrently compute the function value at each point.
- **Step 2**
  As soon as any CPU ends the function evaluation store the result in A.
- **Step 3**
  If A is not full (in the sense that the required N points have not been computed yet), another trial point is chosen over V and start to compute the function value on the same CPU which has just finished to evaluate the function value in the previous point and go to step 2.
- **Step 4**
  Find in A the worst point $\vec{M}$ with function value f_M and the best point $\vec{L}$ with function value f_L.
- **Step 5**
  If the stop criterion (user defined) is satisfied STOP.
- **Step 6**
  Choose randomly $n+1$ distinct points, making our random simplex.
- **Step 7**
  Compute the next trial point $\vec{P}$ using the simplex algorithm by Nelder and Mead [3].
- **Step 8**
  If there is an idle CPU, start to compute the function on this CPU and go to step 6.

- **Step 9**
  Wait for a CPU to end the function evaluation. If the function value in $\vec{P}$ is less than f_M, then the worst point $\vec{M}$ and f_M are replaced by $\vec{P}$ and its function value.
- **Step 10**
  Go to step 4.

A. Implementation on Multivendor CPUs

In order to distribute the computations over a network of workstations, the CRS was designed not by using a parallel compiler, but by using a RPC multi-server network [4]. When it is requested to evaluate a function, the optimizer (the client) asynchronously asks a server to run the simulation. The client process does not wait. The server requests are sent out with a high-level *callrpc()*. So, the client needs to register an **RPC** deamon, running a service itself to catch the replies returned by the servers. To verify the efficiency of the algorithm on distributed CPUs, the same optimization, of which we know the results, was made using up to 10 CPUs on a problem with 8 free parameters. The example deals with a 1-D process simulation with *ssuprem3* [5] for the formation of a bipolar structure (figure 1). The free parameters are related to process formation as implant dose, energy and so on. The CPUs were very different and

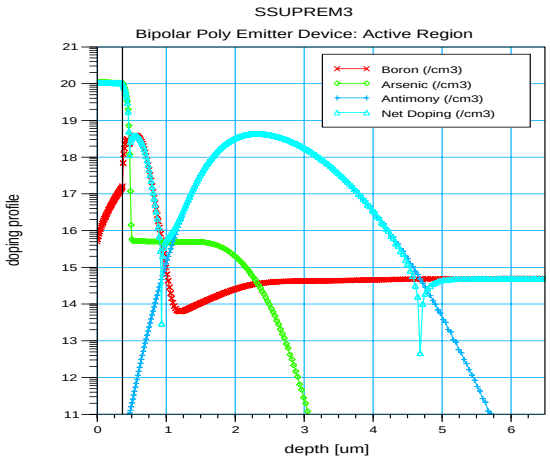

Fig. 1. Bipolar Poly Emitter Device: Active Region

with different loads. We used Sun Sparc10, Sun Sparc20, ULTRASparc, IBM Risc6000 and HP 9000/712 workstations. The final results of the algorithm are summarized in figure 2 and figure 3. We remark that the definition we used of speed-up is not the theoretical one. In fact our workstations have different performance and the speed-up we have used is the ratio to the fastest single WS (in an heterogeneus environment this amounts to a lower bound on the theoretical speed-up). Besides, due to the random aspect of the CRS algorithm, the number of iterations at each test is very different (see figure 2). So, all the algorithm indicators (e.g. the Cost defined as the product of the number of CPU's and the algorithm time, Speed-Up, Efficency defined as the ratio between the Speed-Up and the number of CPU's) was computed per iteration (figure 3).

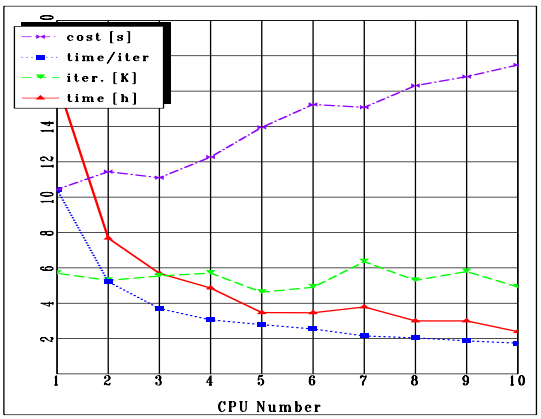

Fig. 2. Cost, Iterations, time per iterations and total time per CPU

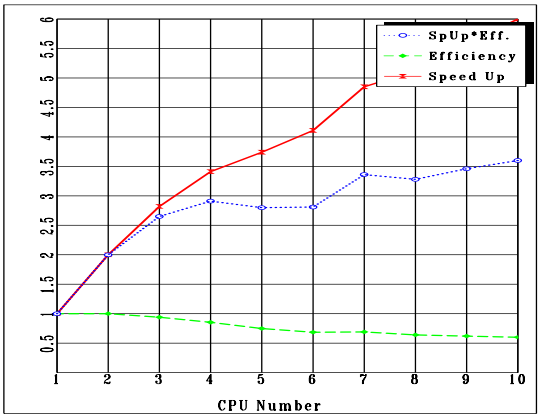

Fig. 3. Speedup, Efficiency and Speedup × Efficiency of the Optimization

IV. Conclusion

In this report a new algorithm was described for global optimization. Even if the new procedure could be slower than those for local optimization, it is well suited for parallel processing so it was implemented to run parallelly on multivendor CPU's. The parallel process was managed by a system RPC multi-server network.

References

[1] W.L. Price, *Global Optimization by Controlled Random Search* Journal of Optimization Theory and Application, vol. 40, N. 3, July 1983

[2] W.L. Price, *Global Optimization Algorithms for a CAD Workstation* Journal of Optimization Theory and Application, vol. 55, N. 1, October 1987

[3] J.A. Nelder and R. Mead, *A simplex method for function minimization* Computer Journal, vol. 7 pp. 308 (1965)

[4] John Bloomer. *Power Programming with RPC* O'Reilly & Associates, Inc., (1992)

[5] SSUPREM3 User's Manual *1D Process Simulation Software* SILVACO International Inc. (1995)