100 Million Dimensions Large-Scale Global Optimization Using Distributed GPU Computing

Alberto Cano and Carlos Garcia-Martinez
Department of Computer Science, Virginia Commonwealth University
Department of Computer Science and Numerical Analysis, University of Cordoba
acano@vcu.edu, cgarcia@uco.es

***Abstract*—At this time, many industrial and science problems deal with a large number of decision variables. Classic metaheuristics have shown excellent search abilities on bounded problems, but they often lose their efficacy when applied to large ones. This is known as the *curse of dimensionality*. To this issue, we have to add the simple fact that the solution evaluation becomes excessively demanding in time. To push the research state forward on this type of problems, the IEEE Congress on Evolutionary Computation regularly organises a competition on large-scale global optimization since 2008. On the other hand, general purpose computing with graphics processing units has become very attractive in the last years, because they may attain very high speed-up ratios on problems with high data parallelism levels. In this work, we study the benefits of exploiting a scalable and distributed computational architecture with multiple GPUs for large scale function optimisation. The study is carried out in terms of 1) evaluation speed-up, 2) quality of the results, and 3) extremely large scale optimisation with real-parameter functions with up to 10^8 variables.**

I. Introduction

Nowadays, it is common to face real-world problems which require optimizing millions of variables [1]. The surge of big data and high-dimensional problems demands algorithms to tackle the optimization of a very high number of dimensions, such as in data mining, DNA and molecular simulation, forecasting, drug discovery, genomic studies, swarm intelligence, etc. Metaheuristics have been successfully applied to many large-scale global optimization problems [2], [3]. However, their performance deteriorates rapidly as the dimensionality of the problem increases, both in terms of the quality of the results and run time [4].

The IEEE Congress on Evolutionary Computation (CEC) has been regularly organising a competition on large-scale global optimization algorithms since 2008 [5], [6]. The last competition benchmark comprises 15 functions with a dimensionality of $1,000$ variables. On most functions, variables cannot be optimised independently to find the global optimum. This interaction is referred to as non-separability. The benchmark functions feature non-uniform subcomponents, imbalance contribution, overlapping subcomponents, ill-conditioning, irregularities, and symmetry breaking, which increase the difficulty of finding the optimum.

The great computing capabilities of modern Graphic Processing Units (GPUs), and the apparition of massively parallel computing frameworks such as CUDA [7], has attracted the research community and promoted the addressing of increasingly larger problems [8], [9], [10], [11], [12]. GPU cards offer programmers a set of multiprocessors with many cores each and a large global memory. These cores can execute in parallel thousands of threads on different sets of data. This type of architecture is especially interesting for problems with high levels of data parallelism.

Recently, Lastra et al. [13] have extended the dimensionality of some functions to high-dimensional problems with 3 million variables, which represents an increase of three orders of magnitude. They employed a single GPU to perform the parallel computation of the functions, thanks to their *additively decomposable characteristic*, i.e., they can be expressed as the sum of other functions. Nevertheless, increasing the dimensionality causes three major problems. First, the run time required to evaluate the high-dimensional functions becomes unmanageable, especially considering that the benchmark is run for 3 million evaluations. Second, the memory available in a single GPU is relatively small, and therefore it limits the dimensionality to a few million variables. Third, the heuristics for high-dimensional problems should be redefined as they do not converge to the optimum as fast as in low dimensional problems.

The objective of this research is to explore the usage of more than one GPU for the task of large-scale function optimisation. For this goal, we first present a several-GPUs-based escalation model of additively decomposable functions, particularly instantiated with the CEC 2013 benchmark functions. We also provide an extension of a memetic algorithm for exploiting the available GPUs when addressing previous functions. Then, we analyse the gained benefits: 1) the speed-up obtained when evaluating a candidate solution; 2) the quality of the generated results with regards to models that do not exploit the existence of GPUs; and 3) the possibility of addressing extremely large-scale problems with up to 10^8 variables, which is a leap of two orders of magnitude as compared with, to our knowledge, the maximum addressed in the literature [13].

This work is structured in the following Sections. First, we describe the approach for several-GPUs-based escalation of additively decomposable functions in Section II. A memetic algorithm taking advantages from the computational power of several GPUs is presented in Section III. The empirical analysis is carried out in Section IV. Finally, Section V depicts the conclusions of the work.

II. Tasks Distribution and Escalation of Additively Decomposable Functions on Multiple GPUs

A function is said to be additively decomposable if it can be defined as a sum of component functions, each of which depends on a subset of the input variables [14]. It turns out that several problems appearing in the literature are additively decomposable, such as NK-landscapes [15], some instances of the quadratic assignment problem [16], or CEC 2013 benchmark functions for the large scale competition [6]. In this Section, we present a task distribution model, of the solution representation and evaluation of additively decomposable functions, for exploiting the presence of multiple GPUs. We firstly describe how we exploit the computational power of a single GPU to carry out the evaluation of P solutions in Section II-A. We comment in Section II-B, how the availability of several GPU cards can be used, the possibilities and drawbacks. Finally, in Section II-C, we propose a escalation model for CEC 2013 functions taking advantages of a distributed architecture with multiple GPUs, and describe a particular example. We shall mention that we similarly exploited the additively decomposable property of these functions to present an extension for the MapReduce paradigm in [17].

A. One GPU Task Distribution

A solution is represented using an array of size D variables. The population of P solutions is represented as an structure of arrays (SoA) to optimize the performance and the access pattern to variables. Indexing of the j-th variable of the i-th solution is computed as $i \cdot D + j$. This access pattern allows to fully coalesce memory accesses and maximizes the memory bandwidth. The main problem is that representing P solutions of size D using double precision variables requires $P \cdot D \cdot 4$ bytes. Consequently, the dimensionality of the functions is limited to a few million variables using a single GPU's memory. Moreover, additional information that the function subcomponents require, such as rotation matrices, are also allocated in the GPU memory.

The processes of initialization and genetic operators are favoured by two levels of parallelism: population parallel and data parallel. The population parallel run concurrent tasks for every solution in P. Data parallel run concurrent tasks for every variable in D. This approach benefits from both levels of parallelism simultaneously. Therefore, initialization, crossover, and mutation run as many threads as $P \cdot D$ to operate in parallel on the j-th variable of the i-th solution.

On the other hand, the evaluation of the solutions demand the majority of the runtime, especially when the dimensionality increases. The evaluation of P solutions with D variables each, according to an additively decomposable function with $NComp$ subcomponents, involves the following stages:

1) Decompose the function evaluation into $NComp$ parallel tasks. Each task may run a unique code for each given subcomponent. If the function does not have subcomponents, there is a single task that involves all the variables from 1 to D.
2) Decompose the subcomponent evaluation into multiple threads. Each task runs a function's code in a subset of variables D_c from y to z according to the function's formula. Multiple threads comprised in thread blocks are scheduled to collaboratively compute the function within their respective range of variables. In turn, each thread eventually computes a fraction of variables D_c/TB according to the threads block size TB and outputs a value that is temporally stored in shared memory. The tasks for each subcomponent are run in parallel using streams, which allows for concurrent data transfer and computing, then maximizing the GPU's occupancy and performance.
3) Reduce the results from the subcomponent decomposition to compute the output for each subcomponent. The partial results allocated in shared memory are reduced (aggregated) to calculate the subcomponent's output (additively decomposable), and they are stored in global memory.
4) Reduce the results from the subcomponent's outputs to compute the final fitness value. After the results for each subcomponent are computed, they are again reduced to calculate the final value as the fitness of the solution. All these stages are all run in parallel for every solution.

An interesting fact is that some variables will need to be read more than once, given that CEC 2013 functions present overlapping components. But nevertheless, this can occur in parallel without any overhead, because the GPU's global memory can be accessed by all its processors and cores.

B. Escalation on Multiple GPUs

The availability of multiple GPUs might enhance previous scheme in two different ways:

- *Time*: If $NComp$ is large enough, the workloads could be reduced by distributing the function subcomponents among the processors of different GPUs. In this case, the presence of overlapping subcomponents would require to duplicate some variables along the memories of the different GPUs, and thus, communication overheads.
- *Dimensionality*: The combined memory of the GPUs can be used to manage a larger number of decision variables. Here it is important to evaluate the time and memory complexity of the function, because the number of cores and available memory are increased just proportionally with the number of GPUs. As an example, functions with $D \times D$ rotation matrices have a quadratic memory complexity, but CEC 2013 functions consider at most 100×100 rotation ones.

In this study, we have opted for scaling up the dimensionality of the functions, by applying the idea introduced by Lastra et al. [13] and Cano et al. [17], which is to replicate the function as a $1,000D$ black box as many times as required. Each replication will be referred to a $1,000D$ *bbox*. The advantages of this methodology are that it avoids communication between GPUs, because the computations in different GPUs are independent, and that memory requirements grow proportionally to the number of replications. On the opposite, it incorporates the disadvantage that function non-separability is restricted to each $1,000D$ group of variables.

In a multi GPU environment with $NGPUs$, there are two main ways to distribute a population of P solutions with a very large dimensionality of D variables. The former is to distribute a subset of the population into each GPU device, then a GPU allocates $P/NGPUs$ solutions with D variables. The former splits the population of P solutions into $NGPUs$. This way, each GPU stores and processes the information of $P/NGPUs$ solutions and all D variables for a given solution are physically stored in a single GPU memory. The latter splits the D variables into $NGPUs$ in a way that each GPU stores and processes the information of $D/NGPUs$ variables, then each GPU allocates a *chunck* of every individual. The main problem with the solutions distribution approach is that it requires a large number of communication between GPUs to perform the genetic operations. Therefore, a large number of memory transfers of size D through the PCI-E bus would significantly impact the performance negatively. On the other hand, the variables distribution approach provides an efficient mapping of the variables and it effectively scales to a very large number of dimensions by simply adding more devices. Genetic operators are not affected since all operations are performed locally, avoiding memory transactions between GPUs. However, when evaluating the quality of the individuals (the fitness function) it is necessary to evaluate each subset of dimensions for each device in parallel, and then synchronize and combine the partial fitness values to compute the final fitness value for the individual, that is communicated to all the devices. Fortunately this memory transaction to the multiple devices is very small, only P float values and therefore its performance impact can be disregarded. The operation of gathering the partial fitness, computing the value and copying back the final fitness is controlled by the main host process.

C. A 10^8 Escalation Example

Consider the evaluation of P 10^8-length solutions on the CEC 2013 $f_4(\vec{x})$ function on 8 GPUs:

$$f_4(\vec{x}) = \sum_{i=1}^{|S|-1} w_i f_{elliptic}(\vec{z_i}) + f_{elliptic}(\vec{z}_{|S|}) \qquad (1)$$

where $S = \{50, 25, 25, 100, 50, 25, 25, 700\}$, $f_{elliptic}(\cdot)$ is a base function, w_i is a weight, and $\vec{z_i}$ is a set of variables obtained from $\vec{x}$, by selecting a subset of size S_i and applying a rotation and a local irregularities introduction transformation. The rotation is not applied for the last set (with $S_i = 700$). Notice that S indicates the subcomponents the function can be additively decomposed into, so the function has exactly $|S|$ subcomponents.

To scale this function up to 10^8 dimensions, we replicate its definition 10^5 times and each GPU is in charge of computing the result of $10^5/8 = 12,500$ bboxes per solution. The data associated with the function is loaded into the GPU, which comprises the weights w_i, the $f_{elliptic}$ definition, the rotation matrices (notice that the largest one is just 100×100), and the information associated with the transformation for introducing local irregularities. Notice as well that this information is loaded just once in the GPUs, and that it is read for each $10^5/8$ bbox processing. We also load the corresponding set of variables of the P 10^8-length solutions into each GPU. This amounts to $P \times 10^8/8$ variables, which requires 2384 MB (single precision) or 4768 MB (double precision) in each GPU if P is set to 100 solutions.

Then, tasks are created for the parallel evaluation of the function. Particularly, each GPU computes $|S| \times P \times 10^5/8$ results, one per function subcomponent, solutions, and function replication. Given that the code associated to each subcomponent is the same, up to $P \times 10^5/8$ tasks could be processed in parallel by threads in the same GPU, so tasks associated to different function subcomponents should be distributed into different processors of the GPU.

The results of the tasks are aggregated at two levels, function subcomponents and function replications, producing P partial values in each GPU. Finally, the real fitness value of each P solution is computed in the host master at the CPU (*central processing unit*) by aggregating their corresponding 8 partial values.

The advantage of this distribution is that it does not require transferring any significant amount of data between GPUs nor to the CPU, then focuses on computationally intensive workload. Moreover, it is highly scalable and allows to address even larger dimensionalities easily by simply adding more GPUs.

III. NGPUs-MA-SW-Chains

NGPUs-MA-SW-Chains is a multiple-GPUs-based extension of the original MA-SW-Chains method [18], which is a steady-state memetic algorithm where only one offspring is produced, and only one solution is optimised, per generation. However, in order to take the most from the computation power of the GPUs, several adaptations have been introduced. Particularly, our model produces in parallel one new offspring per member of the population at the evolution stage, and every member of the population is refined at the optimisation one. We shall mention that, though Lastra et al. already published a single-GPU-based extension of this memetic algorithm [13], ours is better conceived from the original MA-SW-Chains model, and not from this single-GPU one.

Figure 1 presents the global schema of NGPUs-MA-SW-Chains. The key idea is to divide the genotype of the individuals in the population vertically, so each GPU card receives the same portion of all the solutions in the population and

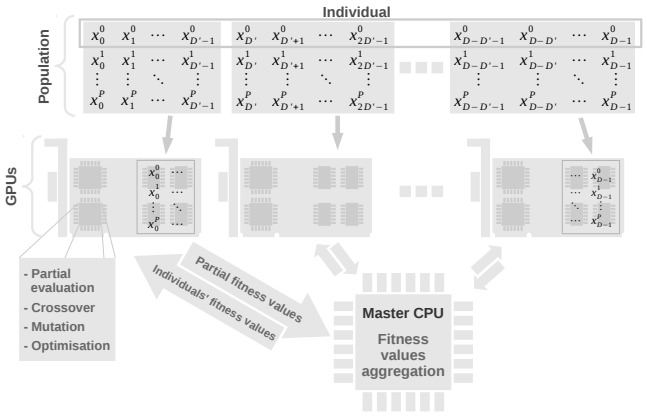

Fig. 1. Global schema of NGPUs-MA-SW-Chains

```
   Input:
    D: Dimensions of the problem
    P: Population size
    Nit: Evol. and opt. iterations
    NGPUs: Number of GPU cards
   Output:
    S: Best solution found
   //Initialisation
1  foreach i ∈ {1,...,NGPU} do //in parallel
2  |  Generate i-th portion of every j-th
   |     solution in P in the i-th GPU;
3  |  f_i^j ←Evaluate i-th portion of every
   |     j-th solution in the i-th GPU;
4  end
5  <Fitness aggregation block; Figure 3>
   //Main loop
6  repeat
7  |  <Pop. evolution block; Figure 4>;
8  |  <Pop. optimisation block; Figure 5>;
9  until Stop-condition is met;
10 return Best-generated-solution();
```

Fig. 2. Pseudocode of NGPUs-MA-SW-Chains

```
   //Evaluation (aggregation part)
1  foreach j ∈ {1,...,NP} do
2  |  F^j ← Aggregate f_i^j, i ∈ {1,...,NGPU}, partial
   |     values;
3  end
4  Broadcast F^j, j ∈ {1,...,NP}, fitness values
      to all GPU;
```

Fig. 3. Additively aggregation of partial function values

applies the evolutionary operations on them. Figures 2 to 5 show the pseudocode of the model. Its operations are:

1) *Population Generation*: each GPU generates randomly a portion of the genotype of the P individuals, according to the box constraints of their respective decision variables (line 2, Figure 2). Concretely, each GPU produces the initial values for those variables involved in its associated $1,000D$ bboxes.

```
1  for Nit iterations do
2  |  foreach i ∈ {1,...,NGPU} do //in parallel
3  |  |  Apply crossover on the i-th portion of
   |  |     every j-th solution in i-th GPU;
4  |  |  Apply mutation on the i-th portion of
   |  |     every j-th solution in i-th GPU;
5  |  |  f_i^j ←Evaluate i-th portion of every
   |  |     j-th solution in the i-th GPU;
6  |  end
7  |  <Fitness aggregation block; Figure 3>;
8  |  foreach i ∈ {1,...,NGPU} do //in parallel
9  |  |  Apply selection in the i-th GPU;
10 |  end
11 end
```

Fig. 4. Pseudocode of the evolution part of NGPUs-MA-SW-Chains

```
1  for Nit iterations do
2  |  foreach i ∈ {1,...,NGPU} do //in parallel
3  |  |  Sample neighbour of i-th portion of
   |  |     every j-th solution in i-th GPU;
4  |  |  f_i^j ←Evaluate i-th portion of every
   |  |     j-th solution in the i-th GPU;
5  |  end
6  |  <Fitness aggregation block; Figure 3>;
7  |  foreach i ∈ {1,...,NGPU} do //in parallel
8  |  |  Apply selection in the i-th GPU;
9  |  end
10 end
```

Fig. 5. Pseudocode of optimisation part of NGPUs-MA-SW-Chains

2) *Evaluation*: each GPU computes the values of its associated $1,000$ bboxes as commented in Section II-B. The partial results produced by each GPU are aggregated within the same card, so each one returns just P partial fitness values to the main host process, one per solution (lines 3, 5, and 4 in Figures 2, 4, and 5, respectively). These partial values are then aggregated by the main host process, obtaining the real fitness value of each individual of the population, and broadcast to the GPU cards (lines 5, 7, and 6 in the respective Figures, and block in Figure 3).
3) *Crossover and mutation*: genetic operators are applied in parallel on the genotype portions that each GPU manages (lines 3 and 4, Figure 4). This is possible because the MA-SW-Chains' genetic operations, BLX-α and BGA [13], have not got any genes interdependence. To take the best from the parallelism, every genotype portion is crossed over with another chosen from the population according to the negative assortative mating [13], which promotes the recombination between distant genotypes (the most distant among three randomly chosen from the population). However, distances computations are restricted to the portion of the genotype associated to each GPU, to avoid communication overheads. Notice that this way, solutions are most frequently crossed over with different solutions at the

same time, each portion with the one of a different chromosome. Subsequently, the resulting genotype portions are mutated.

4) *Selection*: Once offspring have been generated, mutated and evaluated, the P best ones among parents and offspring form the new population (line 9, Figure 4).
5) *Optimization*: Every genotype portion of every individual of the population is optimised in parallel for N_{it} iterations (line 8 in Figure 2, and Figure 5). Solis and Wet's method is considered, as in the original MA-SW-Chains [18]. At each iteration, a new genotype portion is firstly produced and evaluated per solution, from those managed by the GPU; the main process subsequently aggregates the partial fitness values and broadcasts these to the cards; finally, each candidate portion is accepted according to the fitness value of the corresponding whole solution.

IV. Experiments

This section presents the experiments carried out which aim at showing evidence for the following hypothesis:

- First, the task distribution described in Section II allows us to speed up the evaluation of a single solution with regards to the number of available computational units, either single processors or GPU cards (Section IV-A).
- Second, the additional computational resources enable our NGPUs-MA-SW-Chains to generate results better than those obtained by the sequential previous models, MA-SW-Chains [18] and MA-SSW-Chains [19], after the same number of global iterations (Section IV-B). Given that the additional computational resources allow NGPUs-MA-SW-Chains to generate many more solutions per iteration than the other methods, P instead of just 1, the quality improvement should be clearly expected.
- Third, the possibility of using multiple GPUs, the memory increment particularly, capacitates us to address extremely scaled up additively decomposable functions. For this objective, we presents the results of our model on functions with up to 10^8 dimensions with regards to those of a multiple-GPUs-based random search method (Section IV-C).

We have considered the CEC 2013 benchmark [6], which consists of 15 minimization functions, where their global optima have been shifted from the centre of the search space, fitness values are also shifted from 0, and many of them use rotation matrices to create variables interdependences. For each function, we have carried out experiments with 10^3, 10^6, 10^7, and 10^8 variables, by repeating each $1,000D$ bbox as indicated in Section II-B. Ten independent runs were executed per function and dimension.

The parameter setting of NGPUs-MA-SW-Chains is the same as that of MA-SW-Chains, particularly, $P = 100$ for $D = 10^3$ and $D = 10^6$, $P = 50$ for $D = 10^7$, and $P = 25$ for $D = 10^8$, the crossover operator is BLX-α with $\alpha = 0.5$, and $N_{it} = 500$, i.e., the method alternatively applies 500 iterations of the evolutionary scheme and 500 iterations optimising the solutions.

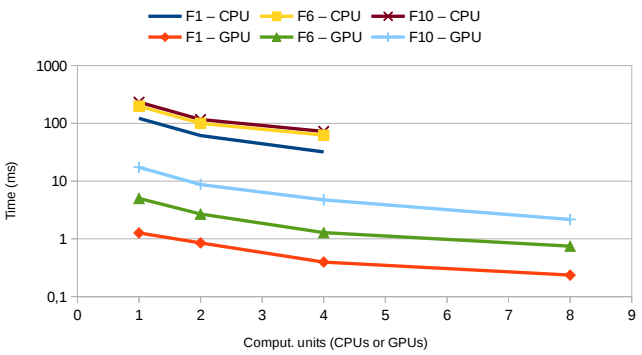

Fig. 6. Time vs number of computing units, either CPUs or GPUs

A. Speed-up Analysis

Figure 6 shows the time required (log plot) to evaluate one 10^6-variables solution on three different functions (F1, F6, and F10), according to the task distribution described in Section II-B and using an increasing number of computing units, either CPUs or GPUs. We observe that:

- The different functions show different complexities, so F1 regularly requires almost half the time of F6 and F10 on the CPUs executions, and a fourth or tenth of F6 and F10 on the GPUs ones, respectively.
- The execution in just one GPU is much faster than in one or several CPUs. This is trivial because a sole GPU card allows us to parallelise the evaluation of the solution on its large number of cores, with minimal communication overheads.
- A larger number of CPUs or GPUs allows more parallelism by distributing the workload and thus speeds up the evaluation. The graphs' curvatures exhibit the communication overheads, i.e., adding more computing units requires further synchronization operations that impedes a lineal gain. However, the overheads due to synchronization are very low as compared to the actual computation time. While CPU paralellism allows to scale up to a very few number of cores in a processor, typically 4, 6 or 8 cores, GPUs comprise thousands of compute cores, and our distribution methodology scales to a potentially high number of GPU devices.

B. Fitness Improvement

Table I shows the mean and standard deviation of the results of NGPUs-MA-SW-Chains, MA-SW-Chains, and MA-SSW-Chains on the different functions, with $D = 1,000$, when they are run along the same number of iterations $(3,000)$. The GPU-MA-SW-Chains model [13] is not included because it was not tested on these functions. Best results are boldfaced. We observe that NGPUs-MA-SW-Chains always obtains the best mean results and smallest deviations in most cases. We carried out a non-parametrical statistical analysis with

TABLE I
NGPUs-MA-SW-Chains vs its Predecessors ($D = 1,000$)

		F1	F2	F3	F4	F5
MA-SW-Chains		6.27e-13	1.24e+3	2.14e+1	4.96e+9	1.86e+6
	Std	5.34e-13	1.42e+2	4.52e-2	2.69e+9	3.66e+5
MA-SSW-Chains		8.18e-13	1.06e+3	2.03e+1	1.28e+10	1.34e+6
	Std	6.88e-13	1.22e+2	3.90e-2	3.79e+9	3.86e+5
NGPUs-MA-SW-Chains		**0**	**4.78e+0**	**2.00e+1**	**9.01e+6**	**1.05e+6**
	Std	**0**	**1.88e+0**	**0**	**7.10e+6**	**1.51e+5**
		F6	F7	F8	F9	F10
MA-SW-Chains		1.01e+6	3.69e+6	4.82e+13	5.38e+8	9.13e+7
	Std	1.38e+4	1.01e+6	9.92e+12	2.34e+8	8.18e+5
MA-SSW-Chains		1.05e+6	8.41e+7	1.44e+14	2.56e+8	9.35e+7
	Std	3.64e+3	2.68e+7	3.18e+13	1.45e+8	**2.38e+5**
NGPUs-MA-SW-Chains		**9.97e+5**	**2.27e+1**	**3.22e+11**	**1.19e+8**	**3.67e+6**
	Std	**1.44e+2**	**1.51e+1**	**3.37e+11**	**1.37e+7**	2.79e+6
		F11	F12	F13	F14	F15
MA-SW-Chains		9.24e+11	1.24e+3	1.89e+7	1.46e+8	5.17e+6
	Std	7.57e+9	**9.69e+1**	2.13e+6	1.75e+7	6.40e+5
MA-SSW-Chains		9.29e+11	1.34e+3	4.92e+9	3.78e+10	8.38e+6
	Std	9.54e+9	1.05e+2	1.63e+9	1.61e+10	1.52e+6
NGPUs-MA-SW-Chains		**1.67e+3**	**4.45e+2**	**1.66e+2**	**3.40e+3**	**5.39e+4**
	Std	**1.02e+3**	2.91e+2	**5.09e+1**	**2.62e+3**	**3.14e+3**

Friedman, Holm and Wilcoxon tests, not shown here for producing trivial results, i.e., there are significant performance differences because NGPUs-MA-SW-Chains always win.

These results reveal that the model is correctly exploiting the computational resources, because it is able to get better results than its predecessors with the same number of generations Notice that this does not necessarily reveal that our approach is better than the others, but just that it can make a good use of the increased numbers of computing units, whereas the others can not because of their sequential nature.

C. Comparison with Random Search

Tables II-IV show the results of NGPUs-MA-SW-Chains on the functions of CEC2013 together with those of a NGPUs-based Random Search approach. The NGPUs-based Random Search exploits the available GPUs to generate in parallel the portions of random solutions, in the same way NGPUs-MA-SW-Chains produces the initial population. Both algorithms are allowed to evaluate a maximum of $500,000$ solutions per execution.

Notice that when the search space is extremely large, and the computational resources limited (few solution evaluations), the necessity of finding relatively good search regions may become much more significant than the need of obtaining the best from the found regions. This means that exploration acquires a significant interest, and any effort invested in exploitation may reduce the performance of the algorithm. Hence, this experiment tests whether the information combination and exploitation carried out by the operations of NGPUs-MA-SW-Chains are still beneficial on these extremely large-scale problems, handling any problem knowledge on real-parameter optimization, with regards to a pure and unconscious exploration approach. Notice that neither MA-SW-Chains, MA-SSW-Chains, nor GPU-MA-SW-Chains can deal with this great number of dimensions and therefore they cannot be compared.

We observe that NGPUs-MA-SW-Chains obtains better mean results on all the functions and dimensions, except

TABLE II
Results in 10^6 dimensions

		F1	F2	F3	F4	F5
NGPUs-MA-SW-Chains		**1.23e+13**	**1.85e+7**	**2.12e+1**	**8.91e+14**	**1.63e+10**
	Std	9.02e+11	4.61e+4	**0**	**3.21e+13**	**7.08e+07**
NGPU Random Search		2.41e+14	5.97e+7	2.17e+1	1.32e+17	7.74e+10
	Std	**3.92e+10**	**9.18e+3**	7.63e-5	9.94e+14	2.11e+8
		F6	F7	F8	F9	F10
NGPUs-MA-SW-Chains		1.08e+9	**2.76e+13**	**1.15e+19**	**1.43e+12**	9.80e+10
	Std	3.77e+4	**2.48e+13**	**8.57e+17**	1.08e+11	1.49e+8
NGPU Random Search		**1.08e+9**	2.34e+21	8.10e+21	5.32e+12	**9.82e+10**
	Std	**2.68e+4**	1.35e+20	2.98e+19	**1.83e+10**	**3.50e+6**
		F11	F12	F13	F14	F15
NGPUs-MA-SW-Chains		**9.72e+15**	**2.82e+13**	**1.26e+16**	**2.92e+16**	**3.50e+16**
	Std	**1.47e+15**	2.21e+12	**3.87e+15**	**2.25e+16**	**2.04e+15**
NGPU Random Search		1.30e+23	1.99e+15	3.74e+22	2.91e+23	1.77e+19
	Std	1.25e+22	**5.76e+11**	2.17e+21	1.84e+22	5.07e+15

TABLE III
Results in 10^7 dimensions

		F1	F2	F3	F4	F5
NGPUs-MA-SW-Chains		**4.68e+14**	**2.72e+8**	**2.14e+1**	**1.66e+17**	**2.71e+11**
	Std	2.75e+13	5.65e+6	4.21e-3	4.61e+15	1.09e+9
NGPU Random Search		2.43e+15	6.00e+8	2.17e+1	1.41e+18	8.00e+11
	Std	**3.62e+11**	**2.55e+4**	**3.55e-15**	**1.37e+15**	**6.06e+8**
		F6	F7	F8	F9	F10
NGPUs-MA-SW-Chains		1.08e+10	**3.02e+17**	**2.19e+21**	**2.22e+13**	9.81e+11
	Std	3.09e+5	**1.57e+17**	2.25e+20	1.22e+12	1.10e+8
NGPU Random Search		**1.08e+10**	8.18e+22	8.67e+22	5.52e+13	**9.81e+11**
	Std	**6.53e+4**	2.03e+21	**1.04e+20**	**6.44e+10**	**1.64e+7**
		F11	F12	F13	F14	F15
NGPUs-MA-SW-Chains		**9.58e+18**	**3.42e+15**	**1.33e+20**	**6.99e+20**	**4.77e+12**
	Std	**3.28e+18**	2.10e+13	**8.35e+19**	**4.50e+20**	**3.77e+11**
NGPU Random Search		9.28e+24	2.00e+16	9.08e+23	1.03e+25	7.05e+16
	Std	5.34e+23	**4.21e+11**	1.65e+22	3.63e+23	5.18e+14

TABLE IV
Results in 10^8 dimensions

		F1	F2	F3	F4	F5
NGPUs-MA-SW-Chains		**8.42e+15**	**1.95e+9**	2.17e+1	**2.44e+18**	**2.55e+12**
	Std	8.65e+14	7.59e+6	8.36e-3	1.73e+17	1.21e+10
NGPU Random Search		2.43e+16	6.00e+9	**2.17e+1**	1.44e+19	8.09e+12
	Std	**2.64e+11**	**4.38e+4**	**0**	**8.24e+15**	**2.91e+9**
		F6	F7	F8	F9	F10
NGPUs-MA-SW-Chains		1.08e+11	**1.81e+19**	**1.95e+23**	**2.07e+14**	**9.74e+12**
	Std	3.42e+6	**4.31e+21**	2.15e+22	1.52e+13	1.22e+10
NGPU Random Search		**1.08e+11**	1.39e+24	8.86e+23	5.59e+14	9.82e+12
	Std	**3.00e+5**	2.51e+22	**5.08e+20**	**7.48e+10**	**5.46e+7**
		F11	F12	F13	F14	F15
NGPUs-MA-SW-Chains		**7.46e+21**	**4.59e+16**	**2.39e+21**	**9.67e+22**	**2.68e+19**
	Std	**1.96e+21**	3.29e+15	**1.33e+21**	**6.16e+22**	1.80e+18
NGPU Random Search		1.70e+26	2.00e+17	1.32e+25	1.56e+26	7.65e+19
	Std	2.43e+24	**4.90e+11**	2.13e+23	1.53e+24	**5.56e+16**

F6 and F10. Therefore, we may conclude that in general, its operations, which combine the information of previous solutions by means of the crossover operator, and refine these solutions by means of the local search, are still favourable in these extremely large-scale contexts. Additionally, we may point out that:

- The structure of F6 and F10 functions must be such that, with these tested dimensions, attempts of exploiting the information of previous solutions (exploitation) is probably less worthy than generating new random solutions (exploration).
- Performance differences between these two algorithms are orders of magnitude larger in some functions (F1, F4, F7, F8 with $D = 10^6$ or $D = 10^7$, F11, F12, F13,

F14, and F15 with $D = 10^6$ or $D = 10^7$), modest in some others (F2, F5, F9, and F15 with $D = 10^8$), and minute in the rest (F3, F6, and F10). Interestingly, we perceive that functions F6 and 10, those where random search gets better results, belong to the group where performance differences are very small, which means that the performance loss for limited exploration is still bounded to reasonable levels.

- Random search usually gets smaller standard deviation results than NGPUs-MA-SW-Chains. This means that random search is more stable producing similar results, although usually inferior, than NGPUs-MA-SW-Chains. Our hypothesis is that the performance of NGPUs-MA-SW-Chains may depend more strongly on the reduced number of search regions explored, so its results vary far more than those of random search from one execution to another.

Table V shows the Wilcoxon-based statistical analysis between the results of NGPUs-MA-SW-Chains and NGPUs-RandomSearch. It shows the sum of rankings of the differences favouring the memetic model, R+, and those for the random search, R-. If the minimum between these two quantities is inferior to the critical value, shown in the third column for 99% confidence factor, then the test finds significant performance differences favouring the algorithm with greater sum of rankings. As it is shown, NGPUs-MA-SW-Chains obtains in general better results than NGPUs-RandomSearch on all the tested dimensions.

TABLE V
WILCOXON TESTS BETWEEN NGPUS-MA-SW-CHAINS (R+) AND NGPUS-RANDOMSEARCH (R-)

Dimension	R+	R-	Critical Value (99%)
10^6	119.5	0.5	15
10^7	118.5	1.5	15
10^8	118.5	1.5	15

To complete the analysis, Figure 7 shows the best, mean and worst intermediate results of NGPUs-MA-SW-Chains, along the search process, on four functions and the four dimensions considered. The functions have been chosen to show different behaviours of the model, depending on the function tackled. We may remark that:

- The algorithm shows a fast approximation towards better fitness values on F1 with $D = 1,000$. However, as the dimension of the problem increases, the method seems to require a warm up period before gaining velocity.
- Regarding F4 and $D = 1,000$, we notice that the optimisation speed is slower than on F1. When the dimension increases, the algorithm seems to find valleys of local optima, hard to scape from.

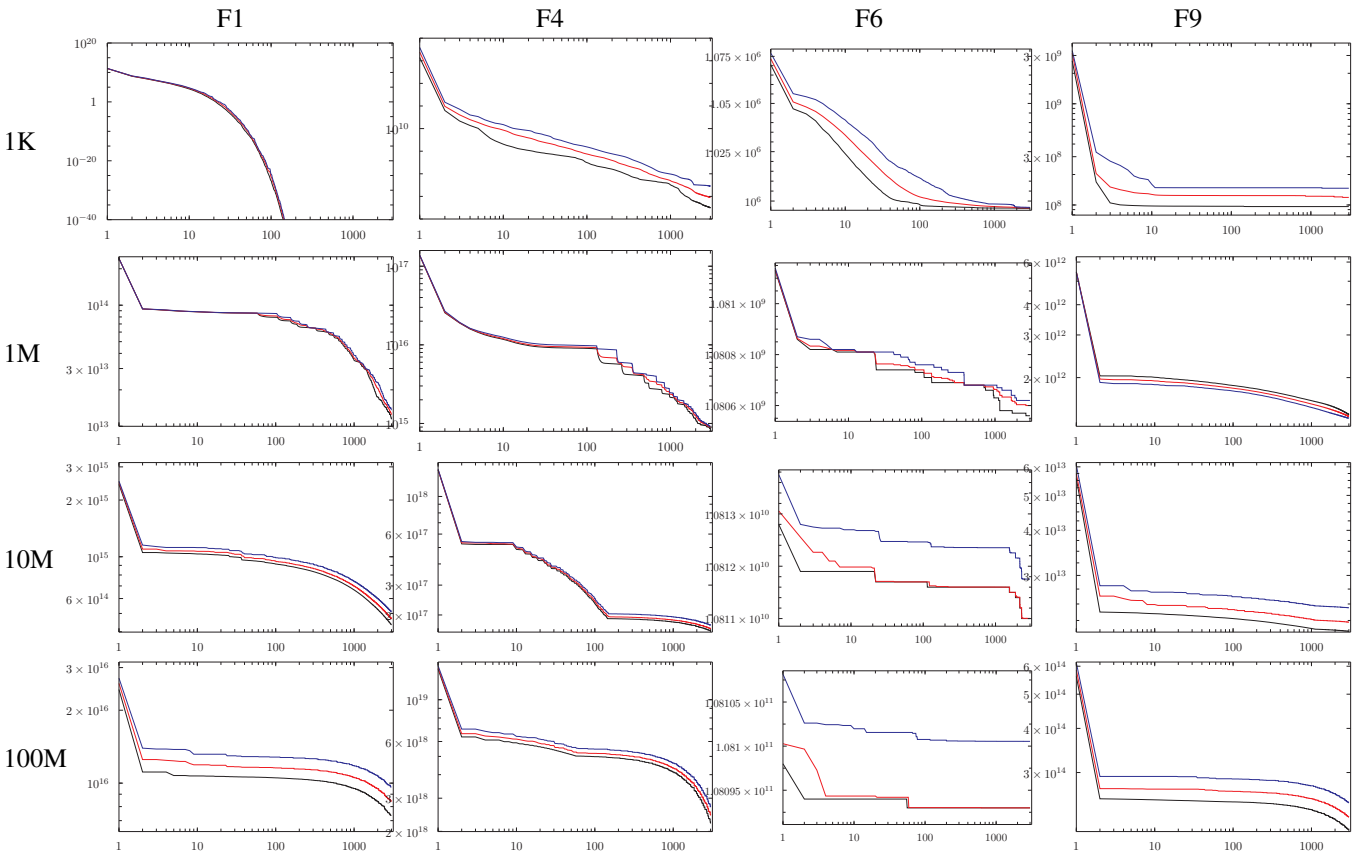

Fig. 7. Convergence graphs of NGPUs-MA-SW-Chains on four functions along the considered dimensions

- On F6, which is one of the functions where random search obtained better results, we observe a normal behaviour when $D = 1,000$, but an unstable one when D is larger. We clearly notice that improvements rarely occur and thus, generating new completely random solutions might be more advantageous.
- Finally on F9, where NGPUs-MA-SW-Chains obtained only modestly better results than random search, the progress seems to be stable, but remarkably slow.

V. Conclusion

In this work, we have addressed the exploitation of a scalable and distributed computational architecture with multiple GPUs for real-parameter large-scale function optimization problems. We have presented an escalation model for the evaluation of additively decomposable functions that allows both, 1) to reduce the time required for the evaluation of candidate solutions and 2) to scale up the problems to an extremely large number of variables, up to 10^8 in this study and two orders of magnitude superior to what have been addressed in the literature, to our knowledge. Additionally, we have adapted a memetic algorithm for taking advantageous of the presence of multiple GPUs. In the empirical study, we present results regarding the solution quality improvement, due to the increased computational capabilities, and the possibility of dealing with the aforementioned 10^8 real-parameter functions.

For future studies, we aim at designing algorithms that further improve the results presented here, and novel escalation models that incorporate the non-separability nature of many functions in the own escalation. Moreover, we also aim at developing an extension of the model to multiple GPUs distributed in several hosts connected through the network. This way we could scale the model and combine the GPUs to address even larger dimensionalities.

Acknowledgment

This work was supported by the Spanish Ministry of Economy and Competitiveness, Project TIN-2014-55252-P, and FEDER funds.

References

[1] S. Shan and G. Wang, "Suvey of modeling and optimization strategies to solve high-dimensional design problems with computationally-expensive black-box functions," *Structural and Multidisciplinary Optimization*, vol. 41, no. 2, pp. 219–241, 2010.

[2] R. Regis, "Evolutionary Programming for High-Dimensional Constrained Expensive Black-Box Optimization Using Radial Basis Functions," *IEEE Transactions on Evolutionary Computation*, vol. 18, no. 3, pp. 326–347, 2014.

[3] A. LaTorre, S. Muelas, and J. Peña, "A comprehensive comparison of large scale global optimizers," *Information Sciences*, vol. 316, pp. 517–549, 2015.

[4] S. Chen, J. Montgomery, and A. Bolufe-Rohler, "Measuring the curse of dimensionality and its effects on particle swarm optimization and differential evolution," *Applied Intelligence*, vol. 42, no. 3, pp. 514–526, 2015.

[5] K. Tang, X. Yao, P. Suganthan, C. MacNish, Y. Chen, C. Chen, and Z. Yang, "Benchmark Functions for the CEC'2008 Special Session and Competition on Large Scale Global Optimization," Technical Report, Nature Inspired Computation and Applications Laboratory, Tech. Rep., 2007.

[6] X. Li, K. Tang, M. N. Omidvar, Z. Yang, and K. Qin, "Benchmark Functions for the CEC'2013 Special Session and Competition on Large-Scale Global Optimization," Evolutionary Computation and Machine Learning Group, RMIT University, Australia, Tech. Rep., 2013.

[7] "NVIDIA CUDA programming and best practices guide." [Online]. Available: http://www.nvidia.com/object/cuda_home_new.html

[8] A. Cano, A. Zafra, and S. Ventura, "A parallel genetic programming algorithm for classification," in *Proceedings of the 6th International Conference on Hybrid Artificial Intelligent Systems (HAIS). Lecture Notes in Computer Science*, vol. 6678 LNAI, no. PART 1, 2011, pp. 172–181.

[9] A. Cano, S. Ventura, and K. Cios, "Scalable CAIM discretization on multiple GPUs using concurrent kernels," *Journal of Supercomputing*, vol. 69, no. 1, pp. 273–292, 2014.

[10] A. Cano, A. Zafra, and S. Ventura, "Speeding up multiple instance learning classification rules on GPUs," *Knowledge and Information Systems*, vol. 44, no. 1, pp. 127–145, 2015.

[11] A. Cano, D. T. Nguyen, S. Ventura, and K. J. Cios, "ur-CAIM: improved CAIM discretization for unbalanced and balanced data," *Soft Computing*, vol. 20, no. 1, pp. 173–188, 2016.

[12] A. Cano, A. Zafra, and S. Ventura, "An interpretable classification rule mining algorithm," *Information Sciences*, vol. 240, pp. 1–20, 2013.

[13] M. Lastra, D. Molina, and J. M. Benítez, "A high performance memetic algorithm for extremely high-dimensional problems," *Information Sciences*, vol. 293, pp. 35–58, 2015.

[14] R. K. Thompson and A. H. Wright, "Additively Decomposable Fitness Functions," Dept. of Computer Science, University of Montana, Tech. Rep., 1996.

[15] S. Kauffman, *The Origins of Order: self organization and selection in evolution*. Oxford University Press, 1993.

[16] M. Drugan, "Instance generator for quadratic assignment problem with additively decomposable cost function," in *Proc. of the IEEE Congress on Evolutionary Computation*, 2013, pp. 2086–2093.

[17] A. Cano, C. García-Martínez, and S. Ventura, "Extremely High-dimensional Optimization with MapReduce: Scaling Functions and Algorithm," *Information Sciences*, pp. 1–31, 2016.

[18] D. Molina, M. Lozano, C. García-Martínez, and F. Herrera, "Memetic algorithms for continuous optimisation based on local search chains," *Evolutionary Computation*, vol. 18, no. 1, pp. 27–63, 2010.

[19] D. Molina, M. Lozano, A. Sánchez, and F. Herrera, "Memetic algorithms based on local search chains for large scale continuous optimisation problems: MA-SSW-Chains," *Soft Computing*, vol. 15, pp. 2201–2220, 2011.