A GPU-based Algorithm for a Faster Hypervolume Contribution Computation

Edgar Manoatl Lopez, Luis Miguel Antonio and Carlos A. Coello Coello*

CINVESTAV-IPN (Evolutionary Computation Group)
Departamento de Computación
México D.F. 07300, MÉXICO
emanoatl@computacion.cs.cinvestav.mx,lmiguel@computacion.cs.cinvestav.mx,
ccoello@cs.cinvestav.mx

Abstract. The hypervolume has become very popular in current multi-objective optimization research. Because of its highly desirable features, it has been used not only as a quality measure for comparing final results of multi-objective evolutionary algorithms (MOEAs), but also as a selection operator (it is, for example, very suitable for *many-objective optimization problems*). However, it has one serious drawback: computing the exact hypervolume is highly costly. The best known algorithms to compute the hypervolume are polynomial in the number of points, but their cost grows exponentially with the number of objectives. This paper proposes a novel approach which, through the use of Graphics Processing Units (GPU), computes in a faster way the hypervolume contribution of a point. We develop a highly parallel implementation of our approach and demonstrate its performance when using it within the $\mathcal{S}$*-Metric Selection Evolutionary Multi-Objective Algorithm* (SMS-EMOA). Our results indicate that our proposed approach is able to achieve a significant speed up (of up to 883x) with respect to its sequential counterpart, which allows us to use SMS-EMOA with exact hypervolume calculations, in problems having up to 9 objective functions.

1 Introduction

Several recent studies have shown that Pareto-based multi-objective evolutionary algorithms (MOEAs) do not perform properly when dealing with problems having more than three objectives (the so-called *many-objective optimization* problems) [12]. This has motivated the development of new selection schemes from which the use of quality assessment indicators is one of the most promising choices. The idea when using this sort of scheme is to maximize a quality assessment indicator that provides a good ordering among sets that represent Pareto approximations. From the many indicators currently available, the hypervolume has been the most popular choice, mainly because it is the only unary

* The third author gratefully acknowledges support from a Cátedra Marcos Moshinsky 2014 and from CONACyT project no. 221551.

quality indicator that is known to be Pareto compliant [20]. The nice mathematical properties of the hypervolume has motivated the development of several hypervolume-based MOEAs (see for example [2, 19]). However, these approaches have a very high computational cost which normally becomes unaffordable for problems having five or more objectives. Although there exist proposals to estimate the hypervolume contribution using sampling (see for example [7]), these approaches are known to have a poor performance with respect to those that use exact hypervolume calculations [13]. Here, we propose a parallel approach, which is implemented in graphics processing units (GPUs), and is coupled to a hypervolume-based MOEA: the *$\mathcal{S}$-Metric Selection Evolutionary Multi-Objective Algorithm* (SMS-EMOA).

The remainder of this paper is organized as follows. Section 2 provides an introduction to the hypervolume, including a short review of the main algorithms that have been proposed to compute it. Our proposed approach is described in Section 3. The experimental results are presented in Section 4, including the methodology and a short discussion of our main findings. Finally, conclusions and some possible paths for future research are provided in Section 5.

2 About the Hypervolume

The hypervolume indicator has become widely used in recent years [18]. This indicator encapsulates in a single unary value a measure of the spread of the solutions along the Pareto front, as well as the distance of the approximation set from the true Pareto optimal front. Whenever one approximation completely dominates another approximation, the hypervolume of the former will be greater than the hypervolume of the latter. Also, the hypervolume is maximized if, and only if, the set of solutions contains all Pareto optimal points. The hypervolume is defined as the n-dimensional space that is contained by an n-dimensional set of points. When applied to multi-objective optimization, the n-dimensional objective values for solutions are treated as points for the computation of such space. That is, the hypervolume is obtained by computing the volume (in objective function space) of the non-dominated set of solutions Q that minimize a MOP. For every solution $i \in Q$, a hypercube v_i is generated with a reference point W and the solution i as its diagonal corner of the hypercube.

$$\mathcal{S} = Vol\left(\bigcup_{i=1}^{|Q|} v_i\right) \tag{1}$$

The hypervolume has important advantages over other set measures [18]:

- It is sensitive to any type of improvements, i.e., whenever an approximation set A dominates another approximation set B, then the hypervolume has a strictly better quality value for the former than for the latter set.
- As a result from the first property, the hypervolume measure guarantees that any approximation set A that achieves the maximally possible quality value for a particular problem contains all Pareto-optimal objective vectors.

– The ranking of the solutions that it generates is invariant to the linear scaling of the objective functions.

In spite of its nice features, the use of the hypervolume is limited by its high computational cost. Hypervolume computation has been proven to be $\#P$-hard (analogous to NP-hard for counting problems) in the number of objectives [3]. As a result, hypervolume algorithms have been used primarily for performance assessment.

Many algorithms have been created to compute hypervolume, each of which has a different worst-case complexity. Next, we introduce the main algorithms that have been proposed for this sake, and we briefly discuss their time complexities.

2.1 Inclusion-Exclusion Algorithm

The Inclusion-Exclusion hypervolume algorithm [17] is perhaps the easiest method for calculating the hypervolume. It works by the inclusion-exclusion principle in the following way: the algorithm adds volumes of rectangular polytopes (n-dimensional rectangular volumes) dominated by each point individually, then subtracts the volumes dominated by intersections of pairs of points; after that, it adds back in volumes dominated by intersections of three points, and so on. Unfortunately, while simple, this method has a time complexity of $\mathcal{O}(n2^m)$ that makes it infeasible on all but the smallest sets.

2.2 LebMeasure Hypervolume Algorithm

The LebMeasure algorithm was proposed by Fleischer [8]. He realized that for any space covered by a set of non-dominated points, one can always identify a rectangular polytope that does not intersect with any other region, so that this region can be lopped off. Then, the hypervolume contributions of these lopped off regions can be easily computed. The hypervolume of the space dominated by these polytopes is then added to a hypervolume accumulator and new points are spawned to reflect the removal of such region. This process can then be repeated until the remaining polytopes no longer dominate any region of space. LebMeasure was initially thought to have polynomial time performance, however it was later demonstrated empirically to exhibit exponential time complexity in the number of objectives and later was proved that the lower bound for LebMeasure's worst case complexity is $\mathcal{O}(2^{n-1})$. Thus, it is also exponential in the number of objectives [15].

2.3 HSO Hypervolume Algorithm

Another hypervolume calculation algorithm is HSO (Hypervolume by Slicing Objectives) [16]. It manages a front by processing one objective at a time and slicing it along the chosen objective. This is known as a *dimension-sweep* algorithm. HSO is given with a front that is pre-sorted with respect to the first

objective. Point values in this objective are used to create cross-sectional slices along this objective. When sweeping along an objective, each point in the list is visited in turn. A list of points is maintained which is sorted in the $(n-1)^{th}$-objective, containing points that have been processed so far, i.e., the points contributing to the current slice. At each slice, there is an $n-1$-objective hypervolume, its hypervolume is calculated recursively and multiplied by the depth of the slice, i.e., the difference between the current point value and the next point value. The point is then added to the $n-1$-objective slice, after removing any points that it dominates. This process is repeated until every point in the list has been visited. While et al. [16] proved that HSO's computational cost is exponential in the number of objectives with a lower bound of $\mathcal{O}(m^{n-1})$, so it still performs poorly for data sets with high dimensionality.

2.4 The FPL Hypervolume Algorithm

The FPL hypervolume algorithm [10] is another dimension-sweep algorithm which improves upon HSO. It adds a new linked data structure which reduces the work required to maintain the fronts built iteratively by HSO. Dominated points must be retained, as points must be reinserted in the reverse order of their deletion. Therefore, dominated points are marked instead of deleted and are skipped over in lower objectives. This data structure improves performance by minimizing the number of comparisons necessary to maintain the sorting within the $n-1$-dimensional slices. It reuses previous calculations when a smaller dimensional slice has already been calculated. Also, hypervolumes are stored along with the current coordinate in the current objective. As these values become staled, bound values which keep track of reusable hypervolumes are updated whenever points are deleted or reinserted. The worst-case complexity of FPL is $\mathcal{O}(m^{n-2}\log m)$. Although all previous described exact hypervolume algorithms and recent ones have led to improved feasibility or better worst-case time complexities, hypervolume calculation remains $\#P$-hard and exponential in the number of objectives [3].

2.5 Hypervolume within MOEAs

The most common way to use the hypervolume as a selection method in MOEAs is through the measure of how much an individual contributes to the hypervolume value of the whole set it belongs to. Then, the solutions that contribute the least to the hypervolume of a front are discarded. The contributing hypervolume of an individual a which belongs to a population P can then be stated in the following way:

$$\mathcal{C}_a = \mathcal{S}(\mathcal{P}, y_{ref}) - \mathcal{S}(\mathcal{P}\backslash\{a\}, y_{ref}) \tag{2}$$

Nowadays, there exist several MOEAs that incorporate the hypervolume in their selection mechanism [19, 2]. However, these approaches have a high computational overload and this creates the necessity of develop alternative strategies

to deal with this problem. Because of this, approximation approaches have also been proposed (e.g., [7] which uses Monte Carlo sampling to approximate expensive hypervolume calculations). Another example is an approach by Bringmann and Friedrich [3] that has a polynomially bounded error and shows promise. These types of methods are faster when the samples are small. However these approaches do not guarantee a bound on the error and most of them deteriorate their behavior as the number of objetives of the problem increases. In fact, in some cases the number of samples needed to produce a good approximation is too large, turning these approaches impractical for many-objective optimization.

3 Proposed Approach

The main idea of our proposed approach is to use all the available hardware resources to calculate the exact contributing hypervolume in a more efficient way, in order to alleviate the high computational overload that current hypervolume-based MOEAs present. Since the computation of the exact hypervolume involves a high computational complexity, in this work we try to circumvent this problem by developing a faster way of computing the exact contributing hypervolume, instead of the hypervolume itself. We propose a way of saving unnecessary hypervolume points computations and a model that is efficient and highly parallelizable. For the descriptions provided next, we assume that we are dealing with sets of non-dominated solutions. Next, we describe our proposed approach, which is implemented in CUDA-C.[1] For the bi-objective case, we take the points of the non-dominated front and sort them in ascending order according to the values of the first objective function f_1. We get then, at the same time, a sequence that is additionally sorted in descending order concerning the f_2 values, because the points are mutually non-dominated. Given a sorted front $SF = \{p_1, \ldots, p_{|SF|}\}$ the contributing hypervolume of a point $C_{\mathcal{P}_i}$ is given by:

$$C_{\mathcal{P}_i} = \Delta f_1 * \Delta f_2 = (p_{i+1,1} - p_{i,1}) * (p_{i,2} - p_{i-1,2}) \tag{3}$$

The graphical representation of this computation can be seen in Figure 2. The parallelization of this two-dimensional approach is done in the following way: using the SIMD[2] model, we work in each thread of the GPU with the computation of the contribution of a point. First, a sorting procedure is performed. For this purpose, we used the so called *bitonic sort* [1], which is a parallel sorting algorithm, originally created for sorting networks. Once the non-dominated front is sorted in ascending order, according to the values of the first objective,

[1] The GPU platform and API developed by Nvidia called CUDA [14] (Computer Unified Device Architecture), which is the one adopted in this work, is based on the CUDA-C language, which is an extension to C that allows development of GPU routines called *kernels*. Each kernel defines instructions that are executed on the GPU by many threads at the same time.

[2] SIMD (Single Instruction Multiple Data) is a computer architecture which can handle only one instruction but applies it to many data streams simultaneously [9].

the i^{th} thread computes the C_{p_i}, so that a set of threads $\mathcal{TH}$ can obtain $C_{\mathcal{P}}$. We consider the case where there might exist more non-dominated points in a set than threads in the GPU and the assignment of the number of threads created in the GPU is done with that in mind. This procedure is presented in Algorithm 1 which shows the architecture of the kernel used for the GPU implementation. The communication between host (CPU) and device (GPU) is done in a synchronous way, since we first need to send the whole set of non-dominated solutions to the device's global memory in order to compute the whole set contributions $C_{\mathcal{P}}$; once the $C_{\mathcal{P}}$ set is ready, it is sent back to the host.

Input: A non-dominated set $\mathcal{P}$ with $\|\mathcal{P}\| = k$, where $\mathcal{P}_i = (p_{i,1}, p_{i,2})$ and a reference point $\mathcal{R} = (r_1, r_2)$
Output: A hypervolumen contribution set $C_{\mathcal{P}}$

Assign an Id for each thread ;
Assign the number of threads created in the GPU to $Dimblock$;
$\mathcal{C}_{\mathcal{P}} \leftarrow 0$;
if $Id = 0$ **then**
  Add the reference point $\mathcal{R}$ to $\mathcal{P}$;
end
$k \leftarrow k + 1$;
Sort in ascending order the set $\mathcal{P}$ in the first objective. $i \leftarrow Id + 1$;
while $i < k$ **do**
  $\mathcal{C}_{\mathcal{P}_i} \leftarrow (p_{i+1,1} - p_{i,1}) * (p_{i,2} - p_{i-1,2})$;
  $i \leftarrow i + Dimblock$;
end
return $\mathcal{C}_{\mathcal{P}}$;

Algorithm 1: Computation of the hypervolumen contribution set $\mathcal{P}$ for two dimensions in a GPU.

For the case of problems with three or more objetives we propose a model in which we try to save unnecessary hypervolume points computations by finding which points in the non-dominated front are not needed for the computation of the contributing hypervolume of an individual. Having a set of non-dominated solutions of μ individuals, the contributing hypervolume of each of the individuals in the whole set can be expressed in the following way:

$$\forall \mathcal{P}_i \in \mathcal{P}, \mathcal{C}_{\mathcal{P}_i} = \mathcal{S}(\mathcal{P}, y_{ref}) - \mathcal{S}(\mathcal{P} \backslash \{\mathcal{P}_i\}, y_{ref}) \tag{4}$$

This means that we will need to compute the hypervolume $\mu + 1$ times. Here, we try to discard the points that are unnecessary for computing the hypervolume contribution of a point, in order to compute volumes of subsets with less dimensions, thus reducing the cost. So, instead of computing each hypervolume contribution $\mathcal{C}_{\mathcal{P}_i}$ with the aforementioned formulation 4, we are able to calculate this contribution $\mathcal{C}_{\mathcal{P}_i}$ in the following way:

$$C_{\mathcal{P}_i} = \prod_{k=0}^{n} (|\mathcal{P}_i[k] - y_{ref}[k]|) - Vol\left(\bigcup_{y \in \mathcal{P}'_i} \{y' | y \prec y' \prec y_{ref}\}\right) \tag{5}$$

where $\mathcal{P}'_i$ is a set of points shifted to be delimited by the non-dominated point $\mathcal{P}_i$. So, we can reformulate the contributing hypervolume as the volume delimited by $\mathcal{P}_i$ and a reference point y_{ref}, which we will call $Vol(\mathcal{P}_i)$, less the volume of the set $\mathcal{P}'_i$, which is a subset of $Vol(\mathcal{P}_i)$, that we will call $Vol(\mathcal{P}'_i)$, also with the same reference point. This idea is presented in Figure 1, where we show a graphical representation of the way in which the hypervolume contribution of a point is computed. The $\mathcal{P}'_i$ set contains the points that we want to find for each non-dominated solution $\mathcal{P}_i$ of a set $\mathcal{P}$, so that we can compute the hypervolume contribution $\mathcal{C}_{\mathcal{P}_i}$ of a point P_i as: $C_{\mathcal{P}_i} = Vol(\mathcal{P}_i) - Vol(\mathcal{P}'_i)$.

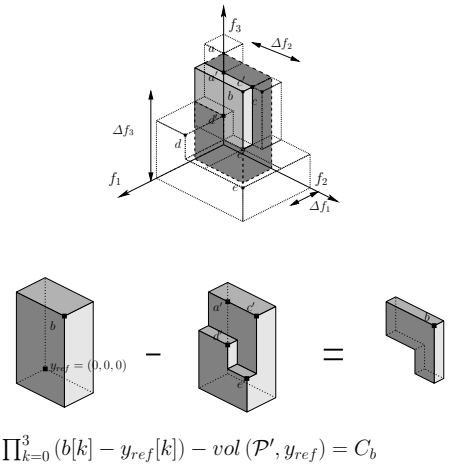

Fig. 1. Computation of the hypervolume contribution of a point b which belongs to a set $\mathcal{P} = \{a, b, c, d, e\}$ of non-dominated solutions with the use of a set of points $\mathcal{P}' = \{a', c', d', e'\}$ delimited by $\mathcal{P}_i$.

This is possible since it is always the case that $Vol(\mathcal{P}'_i) \leq Vol(\mathcal{P}_i)$, because $\mathcal{P}'_i \subset \mathcal{P}$. So, what we want is to cut the volume given by the set of non-dominated solutions $\mathcal{P}$ and the reference point y_{ref}, in order to compute the volume $Vol(\mathcal{P}'_i)$ of reduced sets of points $\mathcal{P}'_i, i = 1, 2, \ldots, |P|$. This is less costly than the way the original aforementioned approach works, for obtaining each individual's contribution. This new $\mathcal{P}'$ set of points, created from a specific non-dominated solution $\mathcal{P}_i$ of a set, might have dominated solutions, so it is necessary to find only the non-dominated solutions of this new set before performing the hypervolume computation of this. The kernel procedure for the computation of the shifted set of points $\mathcal{P}_i$', delimited by a point $\mathcal{P}_i$ of the non-dominated set, is shown in Algorithm 2.

Input: A non-dominated set $\mathcal{P} \subseteq \mathbb{R}^d$ with $\|\mathcal{P}\| = k$, where $\mathcal{P}_i = (p_{i,1}, \cdots, p_{i,k})$ and the current index is i for the point $\mathcal{P}_i$
Output: A set $\mathcal{P}'$

Assign an Id for each thread ;
Assign the number of threads created in the GPU to $Dimblock$;
$Size \leftarrow d * k$;
$i \leftarrow I$;
// Restrict all points which are in the box delimited by $\mathcal{P}_i$
while $i < Size$ **do**
  $l \leftarrow i/d$;
  $m \leftarrow i\%d$;
  if $\mathcal{P}[I][m] > \mathcal{P}[l][m]$ **then**
    $\mathcal{P}'[l][m] \leftarrow \mathcal{P}[l][m]$;
  end
  $i \leftarrow i + Dimblock$;
end
// Filter out all points which are covered by others points
$i \leftarrow Id$;
for $i < k$ **do**
  if $\exists \mathcal{P}'_j \in \mathcal{P}' | \mathcal{P}'_j \preceq^{\ell} \mathcal{P}'_i$ **then**
    Remove $\mathcal{P}'_i$ of $\mathcal{P}'$;
  end
  $i \leftarrow i + Dimblock$;
end
Sort in ascending order the set $\mathcal{P}'$ in the first objective.
return $\mathcal{P}'$;

Algorithm 2: Computation of a set $\mathcal{P}'_i$ for a point $\mathcal{P}_i$.

So, with this idea, we can develop a parallelization of this approach. The model is implemented in two kinds of parallelism: using data-level parallelism[3] and transaction-level parallelism[4] through the use of *streams*[5]. Each *stream* in the whole procedure is responsible for the kernel execution, and for sending a set $\mathcal{P}'_i$ of reduced points solutions from the GPU to the CPU, as well as of the computation of the hypervolume contribution $\mathcal{C}_{\mathcal{P}_i}$ of a point P_i, all of which is done in a parallel way. The number of *streams* adopted depends of each GPU's architecture and on the number of available *streams*. We perform a dynamic task assignation to the *streams* so that the overhead generated is minimal. Figure 3 shows the way the assignation policy is done. The use of *streams* allows us to have an asynchronous communication between the CPU the GPU, since while a $stream_i$ performs the $\mathcal{C}_{\mathcal{P}_2}$ computation, another $stream_j$ executes the *kernel* with its set of parameters and, at the same time, another $stream_l$ downloads data from the GPU.

The overall procedure works in the following way: First, a set of *streams* is created in the host to perform concurrent operations with the GPU. Then, the set of non-dominated solutions $\mathcal{P}$ and the reference point y_{ref} are sent to the

[3] In data-level parallelism, instructions from a single stream operate concurrently on several data.
[4] In a transaction-level parallelism, multiple threads/processes from different transactions can be executed concurrently.
[5] A stream is a sequence of commands, possibly issued by different host threads, that are executed in a certain order.

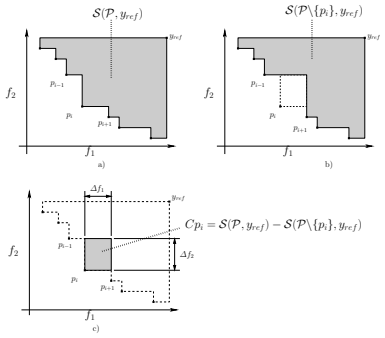

Fig. 2. Contributing hypervolume computation of a point.

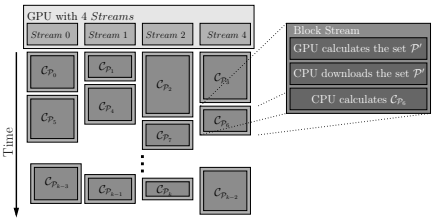

Fig. 3. Task assignation to *Streams*.

GPU. Next, each kernel, executed by an specific *stream*, finds the set $\mathcal{P}'_i$ for a point $\mathcal{P}_i$. Once the set is computed, $\mathcal{P}'_i$ is sent to the host in order to compute the hypervolume contribution $C_{\mathcal{P}_i}$ of such point. Each hypervolume is computed making use of the FPL Hypervolume Algorithm, which is the fastest existent algorithm for the hypervolume computation. The procedure goes on until all the contributions of each point in the non-dominated set had been computed. The pseudocode of the whole procedure is shown in Algorithm 3.

In order to apply our approach, we adopted SMS-EMOA [2], since it fits perfectly with our parallel approach. SMS-EMOA is a steady state evolutionary algorithm in which each newly created solution is ranked and a solution is removed from the worst ranked front in order to maintain the population size. The solution that contributes the least to the hypervolume of the worst ranked front is then discarded (see [2] for details).

4 Experimental results

For purposes of this study, we adopted MOPs from two benchmarks: the Deb-Thiele-Laumanns-Zitzler (DTLZ) test suite [6] (DTLZ2, DTLZ3 and DTLZ4) and the Walking-Fish-Group (WFG) test suite [11] (WFG1, WFG2 and WFG3). We compare our proposed approach with respect to a sequential version of SMS-EMOA which uses the FPL algorithm. In our experiments, we used from 2 to 9 objectives. Our proposed approach is called *$\mathcal{S}$-Metric GPU Selection Evolutionary Multi-Objective Algorithm* (SMGPUS-EMOA).

4.1 Methodology

For our comparative study, we decided to adopt several performance measures as described next.

Input: A non-dominated set $\mathcal{P} \subseteq \mathbb{R}^d$ with $\|\mathcal{P}\| = k$, where $\mathcal{P}_i = (p_{i,1}, \cdots, p_{i,k})$ and a reference point $\mathcal{R} = (r_1, \cdots, r_k)$
Output: The set of hypervolumen contributions $C_{\mathcal{P}}$ of the non-dominated set $\mathcal{P}$

Create s for asynchronous managing of the data ;
Assign an Id for each stream of the GPU;
$\mathcal{C}_{\mathcal{P}} \leftarrow 0$;
if $IdStream = 0$ **then**
  Send the non-dominated set $\mathcal{P}$ to the GPU;
  Send the reference point $\mathcal{R}$ to the GPU;
end
Wait until all the data are sent to the GPU by the CPU;
$IdSig \leftarrow 0$; `// Where IdSig is a shared memory for streams`
`// Start of the transaction parallelism`
for *each stream of the GPU in parallel manner* **do**
  while $IdSig < k$ **do**
    $i \leftarrow IdSig$;
    if $i < NumStreams$ **then**
      $i \leftarrow IdStream$;
    end
    `// Start of the data parallelism`
    Launch the kernel $\mathcal{P}'_i\_computation$ <<<
    $NumSMs/NumStreams, SizeBlock$ >>> $(\mathcal{P}, i, k, d)$;
    Copy the set $\mathcal{P}'_i$ to the CPU ;
    Calculate the $Vol(\mathcal{P}_i)$;
    Calculate the $Vol(\mathcal{P}')$;
    $\mathcal{C}_{\mathcal{P}_i} \leftarrow Vol(\mathcal{P}_i) - Vol(\mathcal{P}'_i)$;
  end
  `// Wait until the shared resource is available`
  while *IdStream is not increased* **do**
    if *IdSig is available* **then**
      $IdSig \leftarrow IdSig + 1$
    end
  end
end
return $\mathcal{C}_{\mathcal{P}}$;

Algorithm 3: Computation of the hypervolumen contribution set $\mathcal{P}$ for three or more dimensions in a GPU.

One of the most important actions in parallel computing is to actually measure how much faster a parallel algorithm runs with respect to the best sequential one. This measure is known as **speedup**. For a problem of size n, the expression for speedup is:

$$S_p = \frac{T_s(n,1)}{T(n,p)} \tag{6}$$

where $T_s(n,1)$ is the time of the best sequential algorithm and $T(n,p)$ is the time of the parallel algorithm with p processors, both solving the same problem.

In order to measure the uniformity of the solutions produced by a MOEA, we adopted **spacing** [4]. This indicator is computed using:

$$\mathcal{S} = \sqrt{\frac{1}{|Q|}\sum_{i=1}^{|Q|}(d_i - \overline{d})^2} \tag{7}$$

where $d_i = \min_{k \in Q \bigwedge k \neq i} \sum_{j=1}^{m} |f_j^i - f_j^k|$ and $\overline{d}$ is the mean value of the above distance measure $\overline{d} = \sum_{i=1}^{|Q|} \frac{d_i}{|Q|}$.

Low values of this indicator reflect a good (uniform) distribution of solutions.

We also analyze its convergence rate with respect to that of the sequential version of SMS-EMOA. For this purpose, we adopted the hypervolume. The reference points used for each of the problems are shown in Table 1. The aim of this study is to identify which of the MOEAs being compared is able to reach the results in a faster way. So, we decided to run each of the MOEAs being analyzed, until reaching a maximum number of function evaluations. At that point, we applied the performance measures previously indicated.

It is worth noting that the sequential version of SMS-EMOA may be unable to achieve the desired number of function evaluations in a reasonable computational time. For that reason, we used an additional stopping criterion: if an algorithm hasn't reached the desired number of objective function evaluations after 8000 minutes, then we stop it. We performed 25 independent runs for each algorithm, problem instance and (given) number of objective functions. The number of objectives used for each problem is shown in Table 2

Problem	**Reference points**
DTLZ1	$(1, 1, 1, \ldots, 1)$
DTLZ2-4	$(2, 2, 2, \ldots, 2)$
WFG1-3	$(3, 5, 7, \ldots, 2m+1)$

Table 1. Reference points used for the hypervolume indicator.

4.2 Parameterization

The parameters of each MOEA used in our study were chosen in such a way that we could do a fair comparison among them. Thus, for both approaches we used the same parameter values since they are similar in everything but the way the hypervolume contribution is computed. The distribution indexes for the SBX and polynomial-based mutation operators [5], used by SMS-EMOA were set as: $\eta_c = 20$ and $\eta_m = 20$, respectively. The crossover probability is $p_c = 0.9$ and the mutation probability is $p_m = 1/L$, where L is the number of decision variables. The internal population size and the maximum number of function evaluations for each problem is defined as indicated in Table 2. For the case of our SMGPUS-EMOA, the number of threads used within the GPU was set in the next way: for problems with two objective functions, 1024 threads were used, with just one stream in the whole process. For the case of three up to 9 objective functions, 1024 threads and seven streams were used. The main characteristics of the hardware used for the experiments are the following: An Intel Core i7-3930k CPU running at 3.20 GHz, with 8GB of RAM 1600 MHz DDR3. Our GPU was a Geforce GTX 680, and we ran our experiments in Fedora 18 (64-bit version).

Problem	Objectives	Population size	Generations	Function evaluations
DTLZ1	2 to 4 and 8	100	250	25000
	5 to 7	95	263	24985
	9	90	277	24930
DTLZ2	2 to 8	100	150	15000
	9	90	166	14940
DTLZ3, WFG1 and WFG3.	2 to 6	100	750	75000
	7 and 8	95	789	74955
	9	90	830	74700
DTLZ4 and WFG2	2 to 7	100	500	50000
	8	110	454	49940
	9	90	555	49950

Table 2. Parameterization values

4.3 Results

Table 3 shows the mean of the spacing and hypervolume values obtained for each final result obtained by the two versions of SMS-EMOA used in our study. Additionally, we show the average time, in minutes, needed to perform the maximum number of function evaluations in each case. When no value is shown in the table for any of the algorithms, this means that it was not able to perform the maximum number of function evaluations after 8000 minutes. The results show that our proposed approach SMGPUS-EMOA is considerably faster and that it achieves a speedup of up to 883x with respect to the sequential algorithm (this speed up is achieved in WFG2). We are also able to obtain the same results as

the sequential version, which verifies that our parallel implementation is working as expected.[6]

5 Conclusions and Future Work

We have proposed a new approach for computing the hypervolume contribution of a point. The core idea of our proposed algorithm is to exploit the parallelization provided by the use of GPUs, combined with a novel implementation that allows us to save unnecesary computations.[7] The proposed algorithm was incorporated within SMS-EMOA, and was tested in several well-known test problems having up to 9 objectives. Our results indicate that our proposed approach is able to achieve a speed up of up to 883x with respect to the sequential version of SMS-EMOA, using the FPL algorithm.

As part of our future work, we would like to incorporate our proposed approach into other hypervolume-based MOEAs (e.g., IBEA [19]). We would also like to develop indicator-based MOEAs that combine the use of the hypervolume with another (less computational intensive) indicator.

References

1. K. E. Batcher. Sorting networks and their applications. In *Proceedings of the April 30–May 2, 1968, Spring Joint Computer Conference*, AFIPS '68 (Spring), pages 307–314, New York, NY, USA, 1968. ACM.
2. N. Beume, B. Naujoks, and M. Emmerich. SMS-EMOA: Multiobjective selection based on dominated hypervolume. *European Journal of Operational Research*, 181(3):1653–1669, 16 September 2007.
3. K. Bringmann and T. Friedrich. Approximating the volume of unions and intersections of high-dimensional geometric objects. *Computational Geometry–Theory and Applications*, 43(6-7):601–610, August 2010.
4. C. A. Coello Coello, G. B. Lamont, and D. A. Van Veldhuizen. *Evolutionary Algorithms for Solving Multi-Objective Problems*. Springer, New York, second edition, September 2007. ISBN 978-0-387-33254-3.
5. K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A Fast and Elitist Multiobjective Genetic Algorithm: NSGA–II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, April 2002.
6. K. Deb, L. Thiele, M. Laumanns, and E. Zitzler. Scalable Test Problems for Evolutionary Multiobjective Optimization. In A. Abraham, L. Jain, and R. Goldberg, editors, *Evolutionary Multiobjective Optimization. Theoretical Advances and Applications*, pages 105–145. Springer, USA, 2005.

[6] There are a few differences in the spacing indicator, which are due to stochastic variations produced by a few isolated runs, which affected the computation of the mean values.

[7] The source code of our proposed approach is available from the first author, upon request.

	SMS-EMOA			SMGPUS-EMOA			
Objectives	Hypervolume	Spacing	Time (mins)	Hypervolume	Spacing	Time (mins)	Speed-up
DTLZ1							
2	**0.873238**	**0.001283**	0.1784	0.873162	0.001457	**0.1354**	1.3186
3	**0.973716**	**0.007665**	0.6921	0.97325	0.009791	**0.5578**	1.2409
4	0.992412	0.017341	9.927	**0.994162**	**0.016266**	**0.9542**	10.404
5	**0.998515**	**0.026013**	146.5268	0.99851	0.02788	**1.9638**	74.6154
6	0.999553	0.038846	1954.4977	**0.99958**	**0.034769**	**9.4954**	205.8373
7	0.999841	0.050285	25632.8153	**0.999876**	**0.041381**	**52.7094**	486.3049
8	–	–	–	**0.99993**	**0.059126**	**437.9792**	–
9	–	–	–	**0.99997**	**0.056958**	**1673.3501**	–
DTLZ2							
2	**3.210879**	0.007702	0.2784	3.210868	**0.007602**	**0.1339**	2.0806
3	**7.426076**	0.041653	1.6382	7.426069	**0.040691**	**0.2926**	5.5995
4	15.5753	0.067581	11.1038	**15.575445**	**0.06661**	**0.673**	16.5008
5	31.677782	**0.084248**	168.761	**31.678164**	0.086913	**1.3908**	121.3476
6	63.75333	**0.133544**	2420.9844	**63.753512**	0.135595	**10.4016**	232.7513
7	–	–	–	**127.79839**	**0.181875**	**49.7891**	–
8	–	–	–	**255.837758**	**0.192845**	**525.0347**	–
9	–	–	–	**511.846819**	**0.168345**	**1713.133**	–
DTLZ3							
2	3.208826	**0.007444**	0.6189	**3.20903**	0.0076	**0.4559**	1.3577
3	7.420476	**0.041679**	1.8316	**7.421543**	0.041999	**1.4572**	1.257
4	**15.574479**	**0.06765**	27.1425	15.57325	0.067938	**2.1452**	12.6527
5	**31.67833**	0.088963	434.3908	31.678284	**0.085466**	**4.2416**	102.4135
6	63.740603	**0.146409**	6040.9012	**63.74412**	0.161621	**23.9932**	251.7766
7	–	–	–	**127.788183**	**0.223037**	**133.1065**	–
8	–	–	–	**255.815976**	**0.192676**	**1021.2014**	–
9	–	–	–	**511.840859**	**0.180436**	**4437.1247**	–
DTLZ4							
2	2.87192	**0.005499**	0.5385	**3.065692**	0.006722	**0.3511**	1.5339
3	6.887928	0.026107	3.1228	**7.145613**	0.032301	**2.5011**	1.2486
4	14.99058	**0.049317**	35.218	**15.406114**	0.057918	**3.4136**	10.3172
5	30.189868	**0.050782**	520.502	**30.787056**	0.062325	**5.6853**	91.5528
6	62.447761	**0.07819**	7446.6796	**63.368461**	0.098488	**26.3014**	283.129
7	–	–	–	**127.726866**	**0.177763**	**125.5394**	–
8	–	–	–	**255.731938**	**0.194006**	**1629.4346**	–
9	–	–	–	**511.694486**	**0.184917**	**4199.4646**	–
WFG1							
2	**6.735278**	**0.006432**	1.6154	6.611425	0.009437	**0.6304**	2.5626
3	**66.595412**	**0.047712**	7.3843	64.496605	0.049191	**4.2319**	1.745
4	550.509191	0.077903	108.9946	**635.368636**	**0.070628**	**5.0127**	21.7439
5	5811.563198	0.077009	2047.7594	**5981.306146**	**0.075655**	**6.6857**	306.2917
6	–	–	–	**69980.43159**	**0.072144**	**28.7109**	–
7	–	–	–	**975295.6509**	**0.074721**	**154.8892**	–
8	–	–	–	**16252223**	**0.071669**	**992.7482**	–
9	–	–	–	**260018586.1**	**0.082601**	**5850.1223**	–
WFG2							
2	**10.655908**	0.008643	0.6955	10.569597	**0.008441**	**0.3571**	1.9476
3	86.215292	**0.04886**	3.7612	**86.827259**	0.052103	**2.4849**	1.5137
4	786.873748	**0.096958**	74.153	**793.125358**	0.101117	**3.3485**	22.1456
5	**8860.244206**	0.082606	1458.6645	8566.72997	**0.07059**	**4.5121**	323.2813
6	110240.8108	**0.100501**	17867.2959	**112357.7782**	0.108096	**20.2233**	883.5017
7	–	–	–	**1646787.124**	0.178076	**134.0804**	–
8	–	–	–	**27973328.67**	0.240691	**1288.5644**	–
WFG3							
2	**10.954985**	0.004739	1.648	10.954594	**0.00423**	**0.6439**	2.5597
3	76.42553	0.132885	6.3563	**76.426786**	**0.132637**	**4.4548**	1.4269
4	683.216698	0.442542	38.2058	**683.589267**	**0.439745**	**7.8809**	4.848
5	7474.344326	**0.704298**	188.9923	**7484.96526**	0.706353	**35.5841**	5.3112
6	96821.66284	0.939095	880.7624	**96950.48865**	**0.905224**	**212.272**	4.1493
7	–	–	–	**1365800.643**	**1.030017**	**779.205**	–
8	–	–	–	**22781648.93**	**1.332752**	**5323.5093**	–

Table 3. Experimental results

7. R. M. Everson, J. E. Fieldsend, and S. Singh. Full Elite Sets for Multi-Objective Optimisation. In I. Parmee, editor, *Proceedings of the Fifth International Conference on Adaptive Computing Design and Manufacture (ACDM 2002)*, volume 5, pages 343–354, University of Exeter, Devon, UK, April 2002. Springer-Verlag.
8. M. Fleischer. The Measure of Pareto Optima. Applications to Multi-objective Metaheuristics. In C. M. Fonseca, P. J. Fleming, E. Zitzler, K. Deb, and L. Thiele, editors, *Evolutionary Multi-Criterion Optimization. Second International Conference, EMO 2003*, pages 519–533, Faro, Portugal, April 2003. Springer. Lecture Notes in Computer Science. Volume 2632.
9. M. J. Flynn. Some computer organizations and their effectiveness. *IEEE Trans. Comput.*, 21(9):948–960, Sept. 1972.
10. C. M. Fonseca, L. Paquete, and M. López-Ibáñez. An Improved Dimension-Sweep Algorithm for the Hypervolume Indicator. In *2006 IEEE Congress on Evolutionary Computation (CEC'2006)*, pages 3973–3979, Vancouver, BC, Canada, July 2006. IEEE.
11. S. Huband, P. Hingston, L. Barone, and L. While. A Review of Multiobjective Test Problems and a Scalable Test Problem Toolkit. *IEEE Transactions on Evolutionary Computation*, 10(5):477–506, October 2006.
12. H. Ishibuchi, N. Tsukamoto, and Y. Nojima. Evolutionary many-objective optimization: A short review. In *2008 Congress on Evolutionary Computation (CEC'2008)*, pages 2424–2431, Hong Kong, June 2008. IEEE Service Center.
13. A. Menchaca-Mendez and C. A. Coello Coello. A New Selection Mechanism Based on Hypervolume and its Locality Property. In *2013 IEEE Congress on Evolutionary Computation (CEC'2013)*, pages 924–931, Cancún, México, 20-23 June 2013. IEEE Press. ISBN 978-1-4799-0454-9.
14. NVIDIA Corporation. Cuda zone, 2014.
15. L. While. A New Analysis of the LebMeasure Algorithm for Calculating Hypervolume. In C. A. Coello Coello, A. Hernández Aguirre, and E. Zitzler, editors, *Evolutionary Multi-Criterion Optimization. Third International Conference, EMO 2005*, pages 326–340, Guanajuato, México, March 2005. Springer. Lecture Notes in Computer Science Vol. 3410.
16. L. While, P. Hingston, L. Barone, and S. Huband. A Faster Algorithm for Calculating Hypervolume. *IEEE Transactions on Evolutionary Computation*, 10(1):29–38, February 2006.
17. J. Wu and S. Azarm. Metrics for Quality Assessment of a Multiobjective Design Optimization Solution Set. *Transactions of the ASME, Journal of Mechanical Design*, 123:18–25, 2001.
18. E. Zitzler, D. Brockhoff, and L. Thiele. The Hypervolume Indicator Revisited: On the Design of Pareto-compliant Indicator Via Weighted Integration. In S. Obayashi, K. Deb, C. Poloni, T. Hiroyasu, and T. Murata, editors, *Evolutionary Multi-Criterion Optimization, 4th International Conference, EMO 2007*, pages 862–876, Matshushima, Japan, March 2007. Springer. Lecture Notes in Computer Science Vol. 4403.
19. E. Zitzler and S. Künzli. Indicator-based Selection in Multiobjective Search. In X. Y. et al., editor, *Parallel Problem Solving from Nature - PPSN VIII*, pages 832–842, Birmingham, UK, September 2004. Springer-Verlag. Lecture Notes in Computer Science Vol. 3242.
20. E. Zitzler, L. Thiele, M. Laumanns, C. M. Fonseca, and V. G. da Fonseca. Performance Assessment of Multiobjective Optimizers: An Analysis and Review. *IEEE Transactions on Evolutionary Computation*, 7(2):117–132, April 2003.