GPGPU-Compatible Archive Based Stochastic Ranking Evolutionary Algorithm (G-ASREA) for Multi-Objective Optimization

Deepak Sharma[1] and Pierre Collet[2]

[1] LogXlabs Research Center, Paris, France
deepak.sharma@logxlabs.com
[2] FDBT, LSIIT, Université de Strasbourg, France
pierre.collet@unistra.fr

Abstract. In this paper, a GPGPU (general purpose graphics processing unit) compatible Archived based Stochastic Ranking Evolutionary Algorithm (G-ASREA) is proposed, that ranks the population with respect to an archive of non-dominated solutions. It reduces the complexity of the deterministic ranking operator from $O(mn^2)$ to $O(man)^\star$ and further speeds up ranking on GPU.

Experiments compare G-ASREA with a CPU version of ASREA and NSGA-II on ZDT test functions for a wide range of population sizes. The results confirm the gain in ranking complexity by showing that on 10K individuals, G-ASREA ranking is $\approx \times 5000$ faster than NSGA-II and $\approx \times 15$ faster than ASREA.

1 Introduction

In the last two decades, the field of multi-objective optimization (MOO) has attracted researchers and practitioners to solve real world optimization problems that involve multiple objectives with a set of solutions known as *Pareto-optimal* solutions. Many optimization algorithms exist in the literature for MOO problems, but the heuristic population based algorithms (also known as multi-objective evolutionary algorithms (MOEAs)) are most suitable to evolve trade-off solutions in one run [1,2].

Though many MOEAs have been developed, there are only a few dominance-ranking based algorithms that are really effective to solve MOO problems. Mostly, these MOEAs differ in their ranking methods which helps to select and propagate good individuals to the next iteration. According to [2], dominance ranking methods can be categorized by dominance rank, dominance count and dominance depth. MOGA [3] and NPGA [4] use a dominance rank method by checking the number of solutions that dominate an individual. NSGA-II [5] sorts individuals according to dominance depth, using the concept of non-dominated sorting [6]. SPEA2 [7] assigns rank, based on dominance depth and dominance count, where the count of dominated individuals by an individual is used. PESA-II [8] is based

* Where, m = nb. of objectives, n = population size and a = archive size.

R. Schaefer et al. (Eds.): PPSN XI, Part II, LNCS 6239, pp. 111–120, 2010.

on dominance count, and uses an archive of non-dominated solutions. All these dominance-based methods evaluate rank serially in a deterministic way (except NPGA), with a quadratic ranking complexity on the population size ($O(mn^2)$ for NSGA-II, where m is the number of objectives and n the population size).

For solving many objective problems [1,2] or Genetic Programming with more than one objective (error minimization and parcimony [9]), a very large population is often required which takes time to both rank and evaluate. However, the advent of General Purpose Graphic Processing Units (GPGPUs) allow to evaluate very large populations in parallel [10]. In [11], the dominance sorting of NSGA-II has been implemented on GPU, but the sorting of individuals in different fronts is still done on CPU. Although this is the first appreciable effort of parallelizing MOEA on GPUs, this paper does attempt to reduce NSGA-II's $O(mn^2)$ ranking complexity.

In [12], we have developed a MOEA called ASREA (Archive based Stochastic Ranking Evolutionary Algorithm) with an $O(man)$ ranking complexity (where a is the size of an archive, that depends on the number of objectives m) which breaks the $O(mn^2)$ complexity, while yielding improved results over NSGA-II.

The present paper shows how ASREA can be fully parallelized, by performing ranking and function evaluation on the GPU. The details of G-ASREA (GPU-based ASREA) are discussed in section 2 followed by experimental results in section 3, in which ranking time and function evaluation of G-ASREA are compared with serial versions of ASREA and NSGA-II. The paper is concluded in section 4.

2 GPGPU Compatible ASREA (G-ASREA)

In [12], the Archive-based Stochastic Ranking Evolutionary Algorithm (ASREA) was shown to converge to the Pareto-optimal front at least as well as NSGA-II and SPEA2, while reaching the front much faster on two-objectives ZDT functions and three-objectives DTLZ functions.

However, the archive updating rule during stochastic ranking is based on CPU serial operations, making the ranking operator difficult to parallelize. This paper shows how it is possible to address this problem, and make ASREA fully GPU-compatible.

2.1 GPGPU

A GPGPU is a General Purpose Graphics Processing Unit, which is a high-performance multithread many-core processor. GPUs have originally been developed for computer graphics, but today's GPUs are capable of performing parallel data computing with support for accessible programming interfaces and industry-standard languages such as C [13]. nVidia GPUs can be programmed using CUDA, which is a general purpose parallel computing architecture, that allows to use the many cores in NVIDIA GPUs to run other algorithms than graphic rendering algorithms. Interested readers may refer to the CUDA programming guide [14].

2.2 Description of the Algorithm

ASREA is an archive-based MOEA that starts with evaluating a random initial population (**ini_pop**). The distinct non-dominated individuals of **ini_pop** are copied to the archive according to an archive updating algorithm 1 described later in this section. As seen in fig. 1, **ini_pop** is copied to **parent_pop** and a rather standard EA loop starts, by checking if a termination criterion is met.

If it is not, a **child_pop** is created by repeatedly selecting *randomly* two parents from the **parent_pop**, and creating children through a crossover, followed by a mutation (in this paper, a standard SBX crossover is used, followed by a polynomial mutation [15]), after which the **child_pop** is evaluated.

Function evaluation on the GPU: Function evaluation (FE) is one of the most time consuming parts of MOEAs and can be easily parallelized on GPUs. In order to perform FE, a single float array X of size vm is allocated in the global memory of the GPU, where v is the number of variables and m is the number of objectives. The variable set of **child_pop** is then copied to X as shown in fig. 2. At the same time, a single float array OBJ of size mn (where n is the child population size) is also allocated in the global memory of the GPU, in order to store the objective function values of **child_pop**, that are computed in parallel by the single instruction multiple data (SIMD) GPU function evaluation kernel.

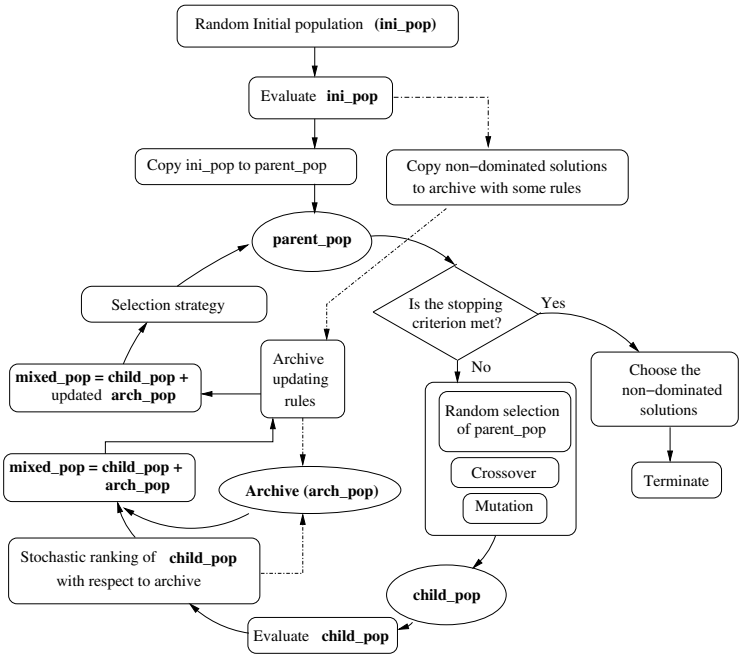

Fig. 1. ASREA flowchart

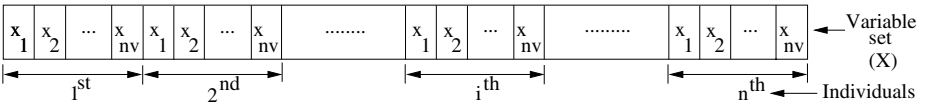

Fig. 2. Single array representation for GPU

Now, comes the time to rank **child_pop**. ASREA's ranking operator [12] reduces the typical ranking complexity of deterministic algorithms such as NSGA-II from $O(mn^2)$ to $O(man)$ (where m is the number of objective, n is the population size and a is the archive size) by comparing individuals of **child_pop** with the members of the archive. The rank of individual (A) of **child_pop** is calculated on the following dominance rank criterion:

$$\text{rank}(A) = 1 + \textit{number of } \textbf{\textit{arch\_pop}} \textit{ members that dominate } A \qquad (1)$$

Note that in ASREA, the lower rank is better, with best rank = 1. The ranking procedure discussed above makes the difference in the working principle of ASREA from other MOEAs, and specially the archived-based ones, in which the archive/parent and current offspring populations are mixed, after which the rank is assigned deterministically [5,7] or every individual of child population is checked for non-domination to become a member of archive [8]. However in ASREA, the archive is used to assign the rank to **child_pop**.

In this paper, the serial portion of ASREA's ranking operator is modified to make it compatible with parallel execution on a GPGPU.

Parallel SIMD ranking on GPU: At this point, *CR*, another integer array of size n (allocated in the global GPU memory at the same time as the *OBJ* array) is used for storing the rank of **child_pop**. Suppose the thread processor *idx* computes the rank of a **child_pop** individual using equation 1, then it stores the calculated rank at position *idx* of *CR*.

During the ranking on GPU, each thread processor also keeps track of the domination count of **arch_pop**, independently. For this, another single integer array AR of size $a \times n$ is allocated in the global GPU memory before execution. Note that the array is initialized to value 1 because all members of **arch_pop** are rank 1 solutions. When an individual of thread processor *idx* dominates the k^{th} member of **arch_pop**, then the thread processor increments $AR[(a \times idx) + k]$ by 1. This dominance check information is used later, to update ranks in the archive.

After parallel stochastic ranking is finished on the GPU, objective function values and ranks of the **child_pop** are updated by copying *OBJ* and *CR* back to the CPU. The rank of the archive is also modified using array AR in the following manner: suppose that the modified rank of the k^{th} member of the archive is evaluated. Then, for $i = 0$ to $n - 1$, the integer value of every $AR[(i \times a) + k]$ is added and finally subtracted by n. If the k^{th} member is still non-dominated,

```
if Number of non-dominated solutions ≤ archive size then
    Copy distinct non-dominated solutions to archive;
else
    Copy the extreme solutions of non-dominated front to archive and evaluate
    crowding distance (CD [5]) of rest of the non-dominated solutions. Fill the
    remaining archive with the sorted CD-wise individuals in descending order;
end
```

Algorithm 1. Archive updating rules

```
Copy the extreme solutions of updated arch_pop to the parent population.
Fill 20% of parent_pop from the updated arch_pop in the following way:
while (20% of parent_pop is not filled) do
    Pick randomly two individuals of updated arch_pop;
    if Crowding distances are different then
        Copy the individual with larger crowding distance into parent_pop;
    else
        Copy any individual randomly;
    end
end
Fill rest of parent_pop from child_pop in the following way:
while (rest of parent_pop is not filled) do
    Pick two individuals randomly from child_pop without replacing them;
    if Ranks are different then
        Copy the individual with smaller rank into parent_pop;
    else
        Copy the individual with larger crowding distance into parent_pop;
    end
end
```

Algorithm 2. Selection strategy to fill the **parent_pop**

then its modified rank is 1. Otherwise, the rank of the k^{th} member depends on the number of **child_pop** individuals who dominated it.

Replacement Strategy. The next step of ASREA is then to update the archive and propagate good individuals to the next generation. First, the ranked **child_pop** and **arch_pop** with modified ranks are mixed together to form **mixed_pop**. Now, the archive is updated from the set of non-dominated solutions ($rank = 1$) of **mixed_pop** as given in algorithm 1.

The next generation (new parent population) is also selected from **mixed_pop** according to the strategy discussed in algorithm 2. Here, 20% of the new parent population is filled from the updated archive with randomly picked members. The rest of **parent_pop** is filled from **child_pop** individuals using tournament selection. The EA loop is then completed and can start again, by checking whether a termination criterion is met (e.g. number of generations) as in fig. 1.

2.3 Salient Features of G-ASREA

The first important feature of G-ASREA comes from its reduced ranking computational complexity $O(man)$ and its implementation on GPGPU for assigning rank to the population by dominance check.

A second important feature of G-ASREA is its inherent ability to preserve diversity in the population. This comes when the not-so-good individuals of **child_pop** may nevertheless obtain a good rank (including rank = 1) because they are only compared with the archive of limited size. Had the ranking method been deterministic, then these not-so-good individuals would not have made it into the next generation and it may have been necessary to implement a diversity preserving scheme in order to avoid premature convergence. This is why ASREA's ranking procedure is *stochastic*.

Another feature of G-ASREA is the drastic but subtle selection strategy that is used to propagate good individuals to the next iteration. Finally, G-ASREA also uses the GPU for function evaluation.

3 Experimental Results

In this paper, five Zitzler-Deb-Thiele functions [16] (summarized in appendix A) are chosen to effectively compare G-ASREA over CPU versions of ASREA and NSGA-II on different population sizes. Every MOEA is run for 25 times using different seeds and average computation time of ranking and function evaluations are shown. Note that G-ASREA's computation time includes copy of data from CPU to GPU, computation on GPU and copy back the data from GPU to CPU. All ZDT functions are executed with identical parameters of MOEAs (cf. table 1) on Intel(R) Core(TM)2 Quad CPU Q8200 @ 2.33GHz computer with one of the 2 GPUs of a GTX295 card.

Algorithms ranking complexity (such as $O(mn^2)$ or $O(man)$) correspond to the *worst* case, which may not appear in a real life. Beyond their theoretical complexity, it is therefore important to assess the complexity of algorithms on real world problems, or at least benchmarks that are supposed to exhibit behaviors encountered in real world problems.

Table. 2 shows the average number of ranking comparisons with NSGA-II, ASREA and G-ASREA for ZDT functions over different population sizes (100 to 100,000). Depending on the problems and their Pareto fronts, different comparison counts are observed for a same algorithm (354.10^6 comparisons for NSGA-II with 10,000 individuals on ZDT1, compared to $3,808.10^6$ comparisons for the same algorithm and same population size on ZDT2).

Table 1. Common parameters for all tested algorithms

No. of generations	100
Individual Xover prob	0.9
Individual Mut prob	1.0
Distrib crossover index	15

Variable Xover prob	0.5
Variable Mut prob	1/(no. of var)
Distrib mutation index	20

Table 2. Real number of ranking comparisons for different population sizes

Problems	ZDT1			ZDT2		
Pop↓ MOEAs→	NSGA-II	ASREA	G-ASREA	NSGA-II	ASREA	G-ASREA
100	1.17 E6	198,463	198,260	968,464	189,059	189,970
1,000	77.8 E6	2.17 E6	2.17 E6	65.2 E6	2.08 E6	2.07 E6
10,000	354 E6	26.2 E6	26.2 E6	3,808 E6	23.9 E6	23.8 E6
100,000	-	413 E6	413 E6	-	346 E6	342 E6

ZDT3			ZDT4			ZDT6		
NSGA-II	ASREA	G-ASREA	NSGA-II	ASREA	G-ASREA	NSGA-II	ASREA	G-ASREA
1.19 E6	198,073	197,940	729,419	181,269	179,710	753,581	191,993	189,120
79.8 E6	2.15 E6	2.15 E6	43.8 E6	1.98 E6	1.98 E6	45.3 E6	2.06 E6	2.05 E6
522 E6	25.5 E6	25.5 E6	2382 E6	22.9 E6	22.9 E6	2395 E6	23.5 E6	23.1 E6
-	392 E6	391 E6	-	330 E6	330 E6	-	372 E6	337 E6

Table 3. Ranking comparison time (in sec.) and speed-up ratio of MOEAs over wide range of population

Problem	ZDT1			Speedup ratio		
Pop↓ MOEAs→	NSGA-II (N)	ASREA (A)	G-ASREA (G)	N/A ratio	N/G ratio	A/G ratio
100	0.000573	0.000101	0.000119	5.6732	4.8151	0.8487
1000	0.037833	0.000999	0.000162	37.8708	233.5370	6.1666
10000	4.304927	0.009971	0.000817	431.7447	5269.1884	12.2044
100000	-	0.098142	0.007011	-	-	13.9983
1000000	-	0.974255	0.065586	-	-	14.8546
Problem	**ZDT2**			**Speedup ratio**		
Pop↓ MOEAs→	NSGA-II (N)	ASREA (A)	G-ASREA (G)	N/A ratio	N/G ratio	A/G ratio
100	0.000481	0.000097	0.000116	4.9587	4.1466	0.8362
1000	0.031627	0.000998	0.000160	31.6903	197.6688	6.2375
10000	3.105234	0.009884	0.000813	314.1677	3819.4760	12.1575
100000	-	0.097470	0.006991	-	-	13.9422
1000000	-	0.972147	0.065659	-	-	14.8059
Problem	**ZDT3**			**Speedup ratio**		
Pop↓ MOEAs→	NSGA-II (N)	ASREA (A)	G-ASREA (G)	N/A ratio	N/G ratio	A/G ratio
100	0.000574	0.000103	0.000121	5.5728	4.7438	0.8512
1000	0.038939	0.001014	0.000162	38.4013	240.3641	6.2592
10000	4.538463	0.009968	0.000834	455.3032	5441.8021	11.9520
100000	-	0.098616	0.007113	-	-	13.8641
1000000	-	0.992453	0.067140	-	-	14.7818
Problem	**ZDT4**			**Speedup ratio**		
Pop↓ MOEAs→	NSGA-II (N)	ASREA (A)	G-ASREA (G)	N/A ratio	N/G ratio	A/G ratio
100	0.000374	0.000094	0.000115	3.9787	3.2521	0.8173
1000	0.020379	0.000952	0.000160	21.4065	127.3687	5.9500
10000	1.159553	0.009277	0.000866	124.9922	1338.9757	10.7124
100000	-	0.092044	0.006947	-	-	13.2492
1000000	-	0.920826	0.066112	-	-	13.9283
Problem	**ZDT6**			**Speedup ratio**		
Pop↓ MOEAs→	NSGA-II (N)	ASREA (A)	G-ASREA (G)	N/A ratio	N/G ratio	A/G ratio
100	0.000372	0.000105	0.000120	3.5428	3.1000	0.8750
1000	0.021340	0.001071	0.000161	19.9253	132.5465	6.6521
10000	1.161053	0.009821	0.000829	118.2214	1400.5464	11.8468
100000	-	0.095291	0.006969	-	-	13.6735
1000000	-	0.947676	0.066241	-	-	14.3064

Significant differences can be seen with ASREA and G-ASREA over NSGA-II, which increase drastically with larger populations, confirming the difference shown by their respective $O(man)$ and $O(mn^2)$ complexities (small differences

are observed between ASREA and G-ASREA because the algorithms were run with different seeds).

ASREA and G-ASREA are further tested for 100,000 individuals (Genetic Programming typical population sizes) where they show a comparable number of ranking comparisons than NSGA-II for 10,000. NSGA-II was not tested for 100,000 individuals over 25 different runs because it took 19.2 hours for just one single run on a ZDT1 function.

ASREA algorithms are not only frugal on comparisons, but G-ASREA can parallelize the ranking (and evaluation) over the GPU processors, allowing for the speedups shown in Table 3, in which the average time of ranking comparison per generation of NSGA-II, ASREA and G-ASREA are given. The N/A and N/G ratios represent speedup of ASREA and G-ASREA over NSGA-II. N/A speedup ranges from 3.54 to 455.3, whereas N/G speedup ranges between 3.1 to 5,441 for 10,000 individuals.

For 10,000 to 1 million[1] population, G-ASREA shows the speedup between 10 to 15 over CPU-ASREA (cf. table 3). The study [11] also showed the same range of speedup in ranking comparison for dominance check on ZDT functions. However, the speedup is limited to this range for two-objective problems because the dominance check of G-ASREA concerns only parallelization of ranking comparisons on the GPU. Nevertheless, if the number of comparisons increases for many objective optimization, then the speedup for dominance check can further increase.

G-ASREA's ranking operator offers a twofold advantage: first, the speedup comes from the smaller ranking complexity over NSGA-II. Then, another advantage comes by being able to perform ranking comparisons on GPU.

Table 4. Function evaluation time (in seconds) of serial and GPU ASREA

Problems	**ZDT6**		Speedup
Pop↓ MOEAs→	ASREA	G-ASREA	ratio
100	0.00053	0.000119	0.4453
1000	0.00521	0.000161	3.2360
10000	0.004798	0.000816	5.8821
100000	0.04728	0.006921	6.8319
1000000	0.4785	0.06526	7.3322

Function evaluation time can also benefit from being run on the GPU. However, this aspect is already studied in several papers [17,11], so only the computation time for ZDT6 is shown in table 3. The speedup is not impressive because the ZDT6 function is not computationally intensive (speedups of up to $\times 250$ have been obtained on GP function evaluations in [10]).

This paper is not so much concerned with solving the ZDT benchmarks (this can be done with hundreds of individuals only [12]) than to study and minimize the bottleneck induced by the ranking operator that may limit the usage of very large populations in multi-objective optimization problems.

[1] Note that only one run is performed for 1 million individuals on each ZDT function.

4 Conclusions and Future Work

The advent of massively parallel GPGPU cards allows to parallelize the evaluation of very large populations in order to solve complex optimization problems. In MOEAs, the bottleneck then becomes the multi-objective ranking stage. In this paper, a GPGPU compatible stochastic ranking operator is proposed that not only requires fewer comparisons for dominance check, but that can parallelize on the GPU card to assign the rank of population. G-ASREA's benefit in ranking complexity is verified as speedups of $\approx \times 5000$ are observed over dominance sorting of NSGA-II on CPU on a population of 10,000 individuals. However, the speedup was limited to $\approx \times 15$ over ASREA for a large population on two-objective problems because the dominance check of G-ASREA concerns only parallelization of ranking comparisons on the GPU.

A future research work will address G-ASREA's clustering method, in order to make it GPGPU compatible so as to obtain further speedups over ASREA and also, to solve many objective problems for which the current crowding distance clustering operator is not very effective. Further investigation is required on the size of archive for many objective problems.

References

1. Deb, K.: Multi-Objective Optimization using Evolutionary Algorithms, 1st edn. Wiley, Chichester (2001)
2. Coello, C.A.C., Lamont, G.B., Veldhuizen, D.A.V.: Evolutionary Algorithms for Solving Multi-Objective Problems. Springer, New York (2007)
3. Fonseca, C.M., Fleming, P.J.: Genetic algorithms for multi-objective optimization: Formulation, discussion, and generalization. In: Proceedings of the Fifth International Conference on Genetic Algorithms, pp. 416–423 (1993)
4. Horn, J., Nafploitis, N., Goldberg, D.E.: A niched Pareto genetic algorithm for multi-objective optimization. In: Proceedings of the First IEEE Conference on Evolutionary Computation, pp. 82–87 (1994)
5. Deb, K., Agrawal, S., Pratap, A., Meyarivan, T.: A fast and elitist multi-objective genetic algorithm: NSGA-II. IEEE Transactions on Evolutionary Computation 6(2), 182–197 (2002)
6. Goldberg, D.E.: Genetic algorithms in search, optimization and machine learning. Addison-Wesley, Reading (1989)
7. Zitzler, E., Laumanns, M., Thiele, L.: SPEA2: Improving the Strength Pareto Evolutionary Algorithm for Multiobjective Optimization. In: Giannakoglou, K., et al. (eds.) Evolutionary Methods for Design, Optimisation and Control with Application to Industrial Problems (EUROGEN 2001), International Center for Numerical Methods in Engineering (CIMNE), pp. 95–100 (2002)
8. Corne, D.W., Jerram, N.R., Knowles, J.D., Oates, M.J.: PESA-II: Region-based selection in evolutionary multiobjective optimization. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCOO 2001), pp. 283–290. Morgan Kaufmann, San Francisco (2001)
9. Baumes, L., Blansch, A., Serna, P., Tchougang, A., Lachiche, N., Collet, P., Corma, A.: Using genetic programming for an advanced performance assessment of industrially relevant heterogeneous catalysts. Materials and Manufacturing Processes 24(3) (March 2009)

10. Maitre, O., Querry, S., Lachiche, N., Collet, P.: Easea parallelization of tree-based genetic programming. In: Congress on Evolutionary Computation (CEC 2010) (2010) (to appear)
11. Wong, M.L.: Parallel multi-objective evolutionary algorithms on graphics processing units. In: GECCO 2009: Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference, pp. 2515–2522. ACM, New York (2009)
12. Sharma, D., Collet, P.: An archived-based stochastic ranking evolutionary algorithm (ASREA) for multi-objective optimization. In: Proceedings of the 12th Annual Conference Genetic and Evolutionary Computation Conference (GECCO 2010), pp. 479–486. ACM, New York (2010)
13. GPGPU: General-purpose computation on graphics hardware, `http://gpgpu.org/`
14. nVidia: nVidia CUDA^{TM} programming guide version 2.3, `http://developer.nvidia.com/object/cuda.html`
15. Deb, K., Agrawal, R.B.: Simulated binary crossover for continuous search space. Complex Systems 9(2), 115–148 (1995)
16. Zitzler, E., Deb, K., Thiele, L.: Comparison of multiobjective evolutionary algorithms: Empirical results. Evolutionary Computation Journal 8(2), 125–148 (2000)
17. Maitre, O., Baumes, L.A., Lachiche, N., Corma, A., Collet, P.: Coarse grain parallelization of evolutionary algorithms on gpgpu cards with easea. In: GECCO 2009: Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation, pp. 1403–1410. ACM, New York (2009)

A Test Functions

Problem definition [16]: **Minimize** $\Gamma(\mathbf{x}) = (f_1(x_1),\ f_2(\mathbf{x}))$; *subjected to:* $f_2(\mathbf{x}) = g(x_2, ..., x_m)h(f_1(x_1), g(x_2, ..., x_m))$, where $\mathbf{x} = (x_1, ..., x_m)$

Functions	Properties
ZDT1: $f_1(x_1) = x_1,\ g(x_2, ..., x_m) = 1 + 9.\sum_{i=2}^{m} x_i/(m-1)$, $h(f_1, g) = 1 - \sqrt{f_1/g}$, where $m = 30$, and $x_i \in [0, 1]$	Convex Pareto-Optimal (P-O) front
ZDT2: $f_1(x_1) = x_1,\ g(x_2, ..., x_m) = 1 + 9.\sum_{i=2}^{m} x_i/(m-1)$, $h(f_1, g) = 1 - (f_1/g)^2$, where $m = 30$, and $x_i \in [0, 1]$	Non-convex P-O front
ZDT3: $f_1(x_1) = x_1,\ g(x_2, ..., x_m) = 1 + 9.\sum_{i=2}^{m} x_i/(m-1)$, $h(f_1, g) = 1 - \sqrt{f_1/g} - (f_1/g)\sin(10\pi f_1)$, where $m = 30$, and $x_i \in [0, 1]$	Discontinuous convex P-O front
ZDT4: $f_1(x_1) = x_1,\ g(x_2, ..., x_m) = 1 + 10(m-1) + \sum_{i=2}^{m}(x_i^2 - 10\cos(4\pi x_i))$, $h(f_1, g) = 1 - \sqrt{f_1/g}$, where $m = 10$, $x_1 \in [0, 1]$, and $x_2, ..., x_m \in [-5, 5]$	21^9 local optimal P-O fronts (Multi-modal)
ZDT6: $f_1(x_1) = 1 - \exp(-4x_1)\sin^6(6\pi x_1)$, $g(x_2, ..., x_m) = 1 + 9.(\sum_{i=2}^{m} x_i/(m-1))^{0.25}$, $h(f_1, g) = 1 - (f_1/g)^2$, where $m = 10$, and $x_i \in [0, 1]$	1. Non-uniform distribution of P-O solutions; 2. Lowest density near the front and highest away from the front