A CUDA Implementation of an Improved Decomposition Based Evolutionary Algorithm for Multi-Objective Optimization

Md Asafuddoula, Hemant Kumar Singh and Tapabrata Ray
School of Engineering and Information Technology, The University of New South Wales, Australia
md.asaf@adfa.edu.au, h.singh@adfa.edu.au, t.ray@adfa.edu.au

ABSTRACT

In last few years, the concept of *decomposition* has been extensively used in a number of evolutionary algorithms, wherein a multi-objective problem is solved as a set of single objective sub-problems. Such algorithms have demonstrated significant break-through for solving problems with four or more objectives (also referred to as *many-objective* optimization problems). Along these lines, authors have previously proposed a decomposition based evolutionary algorithm (DBEA). While DBEA is amenable to parallelization, existing implementations of DBEA (and a number of other such algorithms) use several serial components which are designed for single CPU applications. Recently, parallel computing infrastructure has become increasingly affordable, e.g. graphic processing units (GPUs) and application programming interfaces such as compute unified device architecture (CUDA). Hence, there is a significant interest in the research community to redesign such algorithms to exploit the benefits of parallel computing infrastructure. This work presents an improved CUDA based DBEA. The algorithm aims to offer computational savings via parallelization, while maintaining its performance close to existing state-of-the-art sequential implementations. Parallel structure is deployed for population initialization, evaluation, and selection with preemptive association strategies. Performance of the parallel implementation is presented and compared with its sequential counterpart on a number of well established benchmarks to highlight its benefits.

Keywords

Many-objective Optimization; CUDA; Parallel computation; Decomposition Based Evolutionary Algorithm

1. INTRODUCTION

Over the past decade, evolutionary algorithms based on decomposition [1, 5] shown significant success in solving multi-objective optimization problems. While originally proposed for bi-objective optimization, they have been since extended to deal with problems involving four or more objectives (*many-objective* optimization problems). However, the computational expense involved in solving many-objective problems still remains high. Benchmarking the performance of such algorithms is often non-trivial due to the computational overhead involved in executing them with large population sizes (especially for many-objective problems). While parallelization can offer computational savings, it needs to be ensured that parallelization of the algorithm components does not result in inferior performance. Since such algorithms have several components that are typically designed for serial operation, full benefits of parallelization have not yet been achieved. In recent years, there have been attempts to use the graphics processing units (GPU) for data parallel computation. Affordable desktops with thousands of cores can be deployed for this purpose. In the context of decomposition based algorithms, a GPU implementation of MOEA/D [5] using Ant colony optimization (MOEA/D-ACO) appears in [4]. A GPU based algorithms relying on fast hypervolume computation (SMGPUS-EMOA) was proposed in [3].

This paper presents an improved CUDA based implementation of DBEA (referred to as **cuDBEA**) with its serial components replaced by efficient schemes that exploit parallel computing hardware. These include improved parallel sort via faster CUDA reduction on shared memory, parallel crossover and mutation strategy, and a local dominance based association of the offspring with preferred reference directions. We evaluate the performance of cuDBEA on several numerical benchmarks of the DTLZ test suite and compare it with its sequential implementation (referred to here as seDBEA). The results show significant benefits in terms of computational efficiency without affecting its performance.

1.1 cuDBEA

Similar to I-DBEA [1], cuDBEA uses a set of reference points generated via NBI [2]. cuDBEA initially generates the random states for each individual in parallel and saves them in global memory. The population is then initialized simultaneously using 2D blocks (i.e., number of individuals in x dimension and the number of decision variables in y dimension) of grid and the information is stored in the device global memory. A pseudo-code of the algorithm is presented in Algorithm 1. Computation of extreme point is obtained using faster non-dominated ranking while other components such as computation of ideal point, normalization and computation of distances are similar as described in [1], except cuDBEA utilizes faster CUDA reduction techniques on shared memory. Mating partners for a solution are selected using a roulette wheel selection and offspring are generated via simulated binary crossover and polynomial mutation kernel(Poly) [2].

The association strategy used here is different from the one used in [2], wherein the individuals are associated based on the perpendicular distance from the reference direction to the individual whereas, a theta incorporated local non-domination technique is

GECCO'16 Companion July 20-24, 2016, Denver, CO, USA

ACM ISBN 978-1-4503-4323-7/16/07.

DOI: http://dx.doi.org/10.1145/2908961.2908971

Algorithm 1 cuDBEA

Input: Gen_{max} maximum number of generations, w a set of reference points

1: Initialize the reference points using NBI
2: Generate the random seeds
$<<< NumBlock, NumThread >>>$
3: Initialize population $<<< Grid^*, Block >>>$
4: Evaluate the population
$<<< NumBlock, NumThread >>>$
5: Compute the Ideal point for each objective using CUDA reduction and Threadfence
$<<< NumBlock, NumThread >>>$
6: Compute the Extreme point
$<<< NumBlock, NumThread >>>$
7: **while** $(gen \leq Gen_{max})$ **do**
8: Roulette wheel selection
$<<< NumBlock, NumThread >>>$
9: Generate offspring using SBX
$<<< Grid, Block >>>$
10: Perform polynomial mutation $<<< Grid, Block >>>$
11: Evaluate the offspring
$<<< NumBlock, NumThread >>>$
12: Update Ideal and Extreme points
$<<< NumBlock, NumThread >>>$
13: Associate the offspring to specific direction
$<<< NumBlock, NumThread >>>$
14: Update population $<<< Grid, Block >>>$
15: **end while**
$\{^*NumBlock = \lfloor \frac{NP}{NumThread} \rfloor$, $NumThread = 1024$, $Grid = (\lfloor \frac{D}{BlockSizeD} \rfloor, \lfloor \frac{NP}{BlockSizeNP} \rfloor)$, $Block = (BlockSizeD, BlockSizeNP), BlockSizeD = 32, BlockSizeNP = 32\}$

applied to balance the needs of parallelization versus performance. Solutions within a theta threshold (θ_{th}) are considered as a set of promising individuals (i.e., η_c) that would attempt to associate with a reference direction. θ is computed for an i^{th} individual along the j^{th} reference direction as follows.

$$\theta = cos^{-1}\left(\frac{\mathbf{F^i}.\mathbf{w^j}}{\|\mathbf{F^i}\| \times \|\mathbf{w^j}\|}\right) \quad (1)$$

Local domination is applied to η_c (i.e., a set of promising individuals with a given theta) to identify the non-dominated individuals. The individual with minimum d_2 is then associated with a particular reference direction. Parent individuals are replaced in parallel. If the child solution dominates the parent individual, the parent individual is replaced. In case a child solution is non-dominated with respect to the individuals in the population, it attempts to enter the population via a replacement strategy proposed in [1].

1.2 Numerical experiments

In this section, we analyze the performance of cuDBEA and serial DBEA (seDBEA) based on convergence and computational efficiency using the set of scalable DTLZ problems [2]. Firstly, both algorithms are run for a fixed number of generations (i.e., 1000) on 3-objective DTLZ1, DTLZ2 problems with the population sizes are set as 91, 105, 210, 300, 496, 630 and 1035. The reference directions were generated using the NBI [2] method with sample sizes 12, 13, 19, 23, 30, 34, and 44. Speed-up (S_p) is measured as $S_p = \frac{T_s(n,1)}{T(n,p)}$, where T_s is the computational time of a sequential algorithm for a problem of n tasks by a single processor, $T(n,p)$ is the computational time of a parallel algorithm with p processors. To assess the performance, hypervolume (HV) [6] is used as a performance metric. The results of cuDBEA and seDBEA are shown in Table 1 and the corresponding comparison of computational time is shown in Figure 1. One can see that the results obtained from cuDBEA are better/similar in comparison with seDBEA. The time taken by cuDBEA is significantly less compared to seDBEA, and the difference grows rapidly with the increase in population size.

Table 1: Hypervolume ($Mean_{\text{standard deviation}}$) for DTLZ benchmark problems. Dark grey and light grey backgrounds indicate better mean and standard deviation, respectively.

Prob. (pop-size)	cuDBEA	seDBEA	Ref*.	Speed-up
DTLZ1(91)	$0.78621_{4.7401e-03}$	$0.78476_{1.0306e-02}$	$[0.5023]^3$	1.0521
DTLZ1(105)	$0.78972_{4.9360e-03}$	$0.78842_{6.0686e-03}$	$[0.5015]^3$	1.2159
DTLZ1(210)	$0.80035_{6.2102e-03}$	$0.80020_{6.4234e-03}$	$[0.5002]^3$	2.5092
DTLZ1(300)	$0.80602_{5.9795e-03}$	$0.80458_{1.0495e-02}$	$[0.5002]^3$	3.6174
DTLZ1(496)	$0.98412_{2.2525e-04}$	$0.98381_{8.6038e-04}$	$[1.1380]^3$	5.8569
DTLZ1(630)	$0.95976_{6.8885e-04}$	$0.95906_{2.8329e-03}$	$[0.8311]^3$	7.5521
DTLZ1(1035)	$0.84205_{2.2555e-03}$	$0.83905_{7.3721e-03}$	$[0.5231]^3$	11.931
DTLZ2(91)	$0.40912_{2.1738e-03}$	$0.40978_{2.6031e-03}$	$[1.0000]^3$	0.9866
DTLZ2(105)	$0.41418_{1.8591e-03}$	$0.41460_{2.0694e-03}$	$[1.0000]^3$	1.1354
DTLZ2(210)	$0.43176_{1.6537e-03}$	$0.43058_{2.8611e-03}$	$[1.0000]^3$	2.3533
DTLZ2(300)	$0.43691_{2.3367e-03}$	$0.43745_{2.7414e-03}$	$[1.0000]^3$	3.4451
DTLZ2(496)	$0.44399_{2.6055e-03}$	$0.44388_{2.2481e-03}$	$[1.0000]^3$	5.5362
DTLZ2(630)	$0.45526_{1.5562e-03}$	$0.45439_{1.8484e-03}$	$[1.0043]^3$	7.1969
DTLZ2(1035)	$0.45317_{1.5601e-03}$	$0.45207_{2.1637e-03}$	$[1.0002]^3$	11.564

*Reference point used to compute the hypervolume: Ref = $\mathbf{f}^{\max} = \max([\mathbf{f}_1^{\max}, \mathbf{f}_2^{\max}, ...\mathbf{f}_m^{\max}])$, same for each objective.

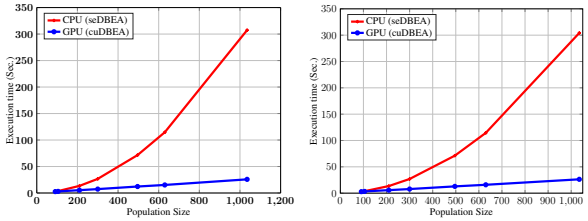

Figure 1: Computational time comparison between GPU (cuDBEA) and CPU (seDBEA) for DTLZ1 and DTLZ2.

1.3 Summary and conclusion

Parallel implementation of DBEA (i.e., cuDBEA) is proposed and compared with its sequential counterpart (seDBEA). It is demonstrated that cuDBEA achieved better or similar performance on the selected test problems, while offering significant savings in computational time. It is also clear that serial implementations do not scale well with increasing population size.

Acknowledgments

The third author would like to acknowledge Future Fellowship and Discovery Project grants from the Australian Research Council (ARC).

2. REFERENCES

[1] M. Asafuddoula, T. Ray, and R. Sarker. A decomposition based evolutionary algorithm for many objective optimization. *IEEE Transactions on Evolutionary Computation*, 19(3):445–460, June 2014.

[2] K. Deb and H. Jain. An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part I: Solving problems with box constraints. *IEEE Transactions on Evolutionary Computation*, 18(4):577–601, 2014.

[3] E. Lopez, L. Antonio, and C. Coello Coello. A gpu-based algorithm for a faster hypervolume contribution computation. In A. Gaspar-Cunha, C. Henggeler Antunes, and C. C. Coello, editors, *Evolutionary Multi-Criterion Optimization*, volume 9019 of *Lecture Notes in Computer Science*, pages 80–94. Springer International Publishing, 2015.

[4] M. Zangari De Souza and A. Ramirez Pozo. A gpu implementation of moea/d-aco for the multiobjective traveling salesman problem. In *Intelligent Systems (BRACIS), 2014 Brazilian Conference on*, pages 324–329, Oct 2014.

[5] Q. Zhang and H. Li. MOEA/D: A multiobjective evolutionary algorithm based on decomposition. *IEEE Transactions on Evolutionary Computation*, 11(6):712–731, 2007.

[6] E. Zitzler, J. Knowles, and L. Thiele. Quality assessment of pareto set approximations. In *Multiobjective Optimization*, volume 5252 of *Lecture Notes in Computer Science*, pages 373–404. 2008.