E-MOGA: A General Purpose Platform for Multi Objective Genetic Algorithm running on CUDA

Sami M. Salamin, Mohammed Aldasht, Hashem Tamimi
sami@ppu.edu, mohammed@ppu.edu, htamimi@ppu.edu
Palestine Polytechnic University

Abstract

This paper introduces an Enhanced Multi Objective Genetic Algorithm (E-MOGA) running on Compute Unified Device Architecture (CUDA) hardware, as a general purpose tool that can solve conflict optimization problems. The tool demonstrates significant speed gains using affordable, scalable and commercially available hardware.

The objectives of this research are: to enhance the general purpose Multi Objective Genetic Algorithm (MOGA) to be ready to execute on CUDA by parallelizing the time consuming parts, to test the performance of the enhanced MOGA on many testing cases, to test the quality of the result of the enhanced MOGA and to study the effect of the number of objectives on performance.

The implemented system works under Matlab environment. It was tested on GPU GeForce GTX 9500 in CUDA platform, showed an average Speedup with more than 28X. The quality, measured using the error with respect to optimal solution, outperform the sequential MOGA.

1 Introduction

Before 2005 GPUs were used only in the video cards for 3D rendering. Their main applications are in personal computers, mobile phones, workstations and embedded systems. The architecture of the GPU differs from CPU; the GPU is designed and specialized for intensive and highly parallel computation involved in graphics rendering [2]. Tens or hundreds of cores (processors), found in the single card devoted to data processing, are related to graphical operations such as rendering, shading... etc, rather than data caching and flow control [2].

The GPUs manufacturers open the way for developers to use their cargo for General-Purpose Computation on Graphics Hardware (GPGPU) by different programming language levels. Compute Unified Device Architecture (CUDA) is one of the most powerful programming tools introduced by Nvidia corporation in 2006. Before that, programming platform such DirectX, OpenGL, HLSL, GLSL were already introduced.

Multi-objective optimization is the process of simultaneously optimizing two or more conflicting objectives subject to certain constraints [5]. The Genetic Algorithm is one of the most popular heuristic techniques and Evolutionary Algorithms to solve Multi-Objective Design and Optimization problems [7].

By returning to many implementations using CUDA with genetic algorithm, one can see different successful applications for this idea in different areas such as: solving theoretical computer science problems including the SATisability application [11]. Biological application such as Autodock [8] (a Drug Discovery Tool), or algorithmic problems such as parallel 0/1 knapsack [12] .

The results show a clear improvement in decreasing running time in many applications by speeding up the system. When we refer

to the Autodock implementations, we can get up to 50x on the fitness function evaluation and 10x-47x speedup on the core genetic algorithm [16].

A very related work can be found in the implementation "CUDA based multi obective parallel genetic algorithm", this work tries to highlight the power of GPU to implement the parallel MOGA in document search problem [6].

2 Backgrounds

2.1 Genetic Algorithms(GA)

The genetic algorithm is a subfield of artificial intelligence that involves combinatorial optimization problems based on heuristic search methods by exploring all the possible solution to get the optimal one (sub-optimal may be sufficient) [3], in many cases it will be a time consuming operation to get the optimal solution.

In computer science, the genetic algorithms are applied in many fields as a searching tool, to find the optimal solution from a very huge solution set [15, 13]. Usually this require us to represent the problem by presenting the different solutions interpreted as a chromosome, such as, an initial population. In addition, mutation, crossover operators, a fitness function and a selection operator for choosing the survivors for the next generation should all be presented [10].

2.2 MOGA

The Multi-Objective optimization also known as multi-criteria or multi-attribute optimization are realistic models for many complex engineering optimization problems. In many real-life problems, the objectives under consideration conflict with each other, and optimizing a particular solution with respect to a single objective can result in unacceptable results with respect to the other objectives. A reasonable solution to a multi-objective problem is to investigate a set of solutions, each of which satisfies the objectives at an acceptable level without being dominated by any other solution. [4].

Genetic Algorithms are one of the most popular heuristic search technique to solve Multi-Objective Design and Optimization problems [7]. When the optimization process applied on a single objective, the result will show one optimal or sub-optimal solution by comparing the solution space and select the best solution. In the case of a vector-valued evaluation function f with $Y \in \Re^k$ and $k > 1$ (k indicate number of objectives), the situation of comparing two solutions x_1 and x_2 is more complex.

In this case the Pareto dominance can be used to find the optimal solution that called the Pareto front. In the Pareto dominance, an objective vector y_1 is said to dominate another objective vectors $y_2 (y_1 \succ y_2)$ if no component of y_1 is smaller than the corresponding component of y_2 and at least one component is greater.

Accordingly, we can say that a solution x_1 is better to another solution x_2, i.e., x_1 dominates x_2, $(x_1 \succ x_2)$, if f(x_1) dominates f(x_2). Here, optimal solutions, i.e., solutions not dominated by any other solution, may be mapped to different objective vectors. In other words: there may exist several optimal objective vectors representing different tradeoffs between the objectives [17].

Let say that we have k objectives, all to be maximized without priority. The solution to this problem can be described in terms of a decision vector $(x_1, x_2, ..., x_n)$ in the decision space $\mathbf{X}$. A function $f : (X \rightarrow Y)$, evaluates the quality of a specific solution by assign-

ing it an objective vector $(y_1, y_2, ..., y_k)$ in the objective space $\mathbf{Y}$ [9] [17].

The set of optimal solutions in the decision space $\mathbf{X}$ is in general denoted as the Pareto set $\mathbf{X}^* \subseteq \mathbf{X}$, and we denote its image in objective space as Pareto front $\mathbf{Y}^* = f(\mathbf{X}^*) \subseteq \mathbf{Y}$.

2.3 CUDA

In order to get the benefit from the power of GPU for general purpose computation, many companies developed an interfacing language to access the complex structure of the GPU card, as a result; There are many programming interface and IDEs for GPU such as HLSL, GLSL, Cg or brookGPU, but the most common used and advanced is CUDA (Compute Unified Device Architecture).

In November 2006, NVIDIA introduced CUDA [14]. a general purpose parallel computing architecture, with a new parallel programming model and instruction set architecture, which uses the parallel compute engine in NVIDIA GPUs to solve many complex computational problems in a more efficient way than on a CPU. CUDA comes with a software environment that allows developers to use C as a high-level programming language [2].

However CUDA works only within environments that contain NVIDIA graphic cards in it [1]. By programming under CUDA, the operations can be addressed to either GPU or CPU.

By using CUDA, the CUDA-enabled GPU (called device) is exposed to the CPU (called host) as a co-processor. This means that each GPU is considered to have its own memory and processing elements that are separated from the host computer. To perform useful work, data must be transferred between the memory space of the host computer and CUDA device(s). For this reason, performance results must include input and output (IO) time to be informative.

3 E-MOGA paltform

The designed platform can work on any computing environment contains the CUDA enabled device and a pre-installed software. The needed software are: CUDA, Matlab, GPUmat, Visual C++.

CUDA architecture is designed to execute an arithmetic or logic operation on huge amount of data. On the other hand, this architecture is not designed for looping, branching or if-statement instruction.

Since the Multi-Objective Genetic Algorithm is designed and based on many loops, branching, sorting and computation; the implementation phase will split the program into two parts of sub-programs:

1. Sub-program that runs on GPU; high computation without the looping or branching style.

2. Sub-program that runs on CPU; the main subprogram, all the branching, looping and conditional (the main part).

To convert MOGA into E-MOGA, many modification should be take a place. The modifications can be summarized as follow:

1. Transferring the huge mathematical computations to the GPU instead of CPU.

2. Exploring the loops as vectors; parallelizing loops.

3. Mixed implementation; C/C++ and Matlab script.

4. Convert from Matlab script to compiled files.

The modifications that summarized in the previous points, tries to increase the performance of the system; in the first point, it transfer the huge computation after parallelizing its operation to be executed on GPU instead of CPU. The second modification to parallelizing the loops and the conditional statements by replace it with parallel exploration. The third point is to use a mix implementation between different languages to get integrate between performance and ease of use. The last on is to convert the Matlab script to a compiled file, this will reduce the execution time.

To realize the objective of our research we propose the structure shown in figure 1, the structure show the relation between system component as layers. In this figure, it's shown that the E-MOGA will split the execution between CPU and GPU. The Mtlab will host the E-MOGA execution, and as we know, the Mtlab is not designed to be executed on GPU, so, additional layers were used to make the Matlab able to execute on GPU. The additional layer consists from two SDK, C for CUDA and GPUmat. GPUmat is a Software Development Kite that allows standard MATLAB code to run on GPU.

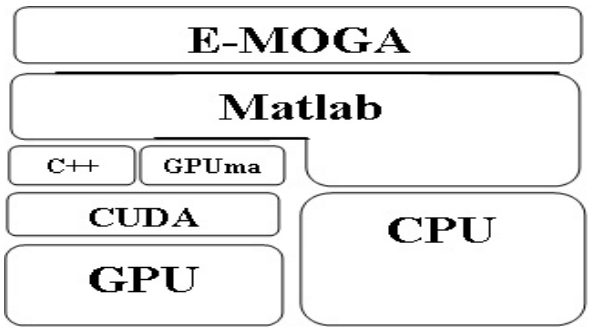

Figure 1: System structure

Table 1: benchmarks mathematical formulation

benchmark	mathematical formula
ZDT1	$f_1(x) = x_1$ $g(x) = 1 + \frac{9}{n-1}\sum_{i=2}^{n} x_i$ $h(f_1, g) = 1 - \sqrt{\frac{f_1}{g}}$
ZDT2	$f_1(x) = x_1$ $g(x) = 1 + \frac{9}{n-1}\sum_{i=2}^{n} x_i$ $h(f_1, g) = 1 - (\frac{f_1}{g})^2$
ZDT3	$f_1(x) = x_1$ $g(x) = 1 + \frac{9}{n-1}\sum_{i=2}^{n} x_i$ $h(f_1, g) = 1 - \sqrt{\frac{f_1}{g}} - \frac{f_1}{g} \times \sin(10\pi f_1)$
ZDT4	$f_1(x) = x_1$ $g(x) = 1 + 10 \times (n-1) + \sum_{i=2}^{n} \left[x_i^2 - 10 \times cos(4\pi x_i))\right]$ $h(f_1, g) = g(x) \left[1 - \sqrt{\frac{f_1}{g(x)}} - \frac{f_1}{g(x)} sin(10\pi f_1)\right]$
ZDT6	$f_1(x) = 1 - exp(-4x) sin^6(6\pi x_1)$ $g(x) = 1 + 9\left[\frac{\sum_{i=2}^{n} x_i}{n-1}\right]^{0.25}$ $h(f_1, g) = g(x)\left[1 - (\frac{f(x_1)}{g(x)})^2\right]$
Schaffer F6	$f(x_1, x_2, ..., x_n) = 0.5 + \frac{\sin^2(\sqrt{\sum_{i=1}^{n} x^2}))}{1+0.001\times(\sum_{i=1}^{n} x^2)^2}$
Griewank	$f(x1, x2, ..., xn) = 1 + (0.00025) \times \sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} \cos(\frac{x_i}{\sqrt{i}})$

4 Experimental Results and Analysis

this section contain the experimental tests and results.

4.1 Experimental Results

The obtained results of the enhanced version of MOGA were analyzed, tested and compared to the performance of the existing system to ensure the quality of the results. The tests steps are done on a selected benchmarks. Table 1 shows the selected benchmarks with the used constrains.

To make the testing process easy and fair, the selected regions(constrains), dimensionality and the optimal solution vector were set the same for all benchmarks as shown in table 2.

MOGA needs to define the stopping criteria. The used stopping criterias are: A population of 15*(the number of variables) individuals and maximum generations of 400*(the number of variables) with epsilon 10e-3 (the

Table 2: benchmarks properties

property	value
constrains	$-100 \leq x_i \leq 100$
dimensionality	$10, 20, 30, 40, 50$
optimal solution	$x^* = \{0, ..., 0\}$

difference between the change of good solutions).

4.2 Performance Analysis

Depending on the stopping criteria, and, since MOGA will start the population randomly, so the algorithm will not take a fixed time to reach the Pareto set for any benchmark each run time, so the average time should be taken for (n) executions.

The tests were taken for both systems; CPU only and CPU-GPU with different number of objectives on each benchmark. The tests results are shown in table 3:

Table 3: execution time for benchmarks on the existing and proposed systems

Benchmark	Dimension	CPU-time	GPU-time	SpeedUp
ZDT1	10	217	77	3
	20	785	178	5
	30	3218	224	15
	40	4233	269	16
	50	6281	340	19
ZDT2	10	379	56	7
	20	548	69	8
	30	3177	210	16
	40	4216	239	18
	50	4979	197	26
ZDT3	10	504	135	4
	20	1085	144	8
	30	3129	188	17
	40	4238	214	20
	50	14035	260	54
ZDT4	10	133	46	3
	20	970	133	8
	30	1190	150	9
	40	2318	182	13
	50	8263	227	37
ZDT6	10	123	32	4
	20	382	68	6
	30	1010	113	9
	40	2118	159	14
	50	9820	214	46
Schaffer F6	10	100	32	4
	20	197	60	4
	30	572	100	6
	40	1331	167	8
	50	1413	162	9
Grewink	10	77	28	3
	20	363	61	6
	30	874	97	9
	40	1600	158	11
	50	9820	214	46

4.3 Solutions Quality Analysis

The quality is the error measure in the results, and it is an important factor that should not be ignored. To check the quality of the results, we can graphically compare between the outputs of the two systems by drawing the results of any two variables (2-D) and make a comparison between the points positions. In Figure 3, a comparison between two outputs are shown.

It's clear that both figures almost contain the same number of points, but a deep look to the CUDA result, the result distribution is very close to the optimal solution (the optimal solution (0,0) in this case), on the other hand, the CPU solution take a larger range of distribution.

The graphical comparison is not an enough indicator of the results quality, so, a

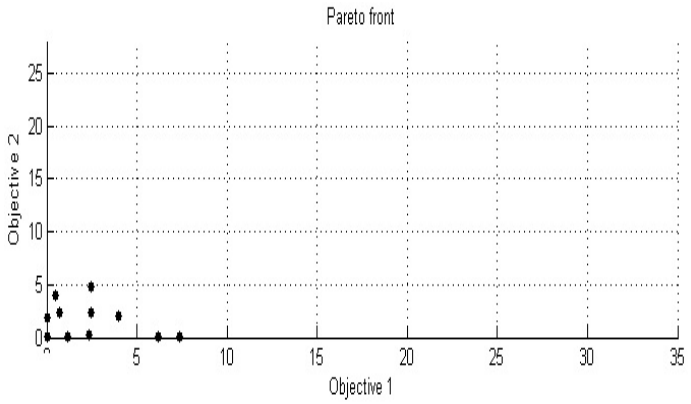

Figure 2: the Pareto front after the final iteration on E-MOGA

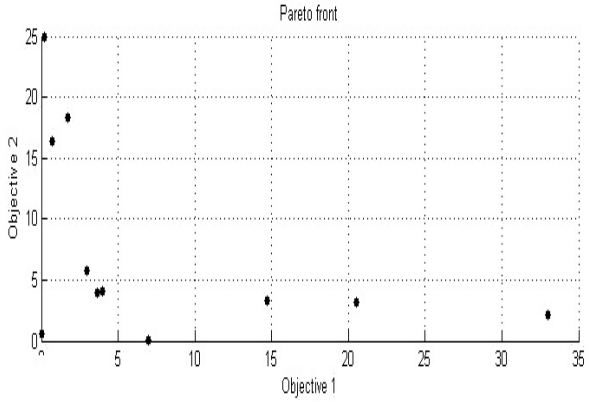

Figure 3: the Pareto front after the final iteration on MOGA

new idea was implemented in equation 1.

$$Q = \sqrt{\sum_{i=1}^{n} (\mathbf{X}_i - \mathbf{X}^*)^2}. \qquad (1)$$

The quality can be measured by taking the summation of distances between the Pareto front and the optimal solution; the smallest value indicates that the Pareto front includes more feasible solutions. Table 4 shows a sample of a scalar values that indicate the results quality:

Table 4: solution quality for benchmarks on the exiting and proposed systems

Benchmark	Dimension	CPU-Quality	GPU-Quality	Quality difference
ZDT1	10	64.5	28.5	36
	20	112.9	34.9	78
	30	202.3	69.6	132.7
	40	164.2	69.9	94.3
	50	186	97.8	88.2
ZDT2	10	408	17.7	390.4
	20	390.4	115.8	274.7
	30	573.9	54.7	519.1
	40	613.3	74.8	538.5
	50	152.9	93	59.9
ZDT3	10	201.2	119.2	82
	20	69.1	41.3	27.9
	30	299.1	58.5	240.7
	40	337.5	94.2	243.3
	50	167.6	102.7	64.9
ZDT4	10	121.8	16.2	105.6
	20	81	30.3	50.7
	30	53.5	49.4	4.1
	40	98.6	74.2	24.3
	50	98.8	83.8	15
ZDT6	10	27.2	22.7	4.5
	20	122.2	45	77.1
	30	224.5	64.3	160.2
	40	105.5	86.6	18.8
	50	489.9	108.9	381
Schaffer F6	10	24.1	4	20.1
	20	58.4	5	53.4
	30	81.6	8	73.6
	40	83.6	7	76.6
	50	129.7	11	118.7
Grewink	10	17	3	14
	20	36	5	31.0094
	30	64.4	6	58.4
	40	86	8	78

5 Analysis

From table 3 it's clearly shown that the proposed E-MOGA is faster than the exiting MOGA in all cases and when the dimensionality (number of variables) increases, the power of the GPU raises to handle the complex computation on its parallel architecture.

The speed up varies due to the complexity of the problem; as the number of objectives increase, the system becomes more complex to be solved and needs more time. The GPU performance can beat the CPU performance all the time. The minimum speed up can't be less than 2.7X and the speed up can increase to become more than 53X in some cases. The following figure 4 shows the relation between the number of objective and speed up.

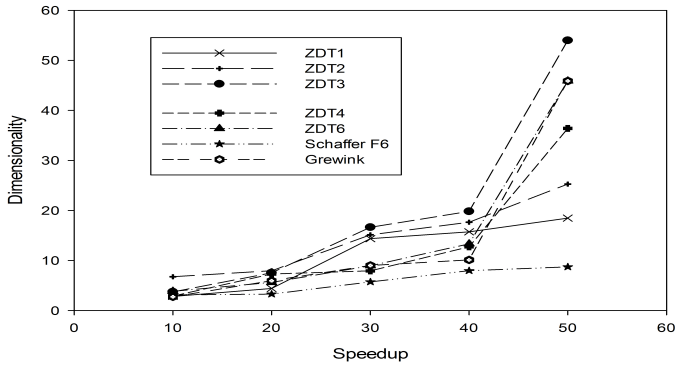

Figure 4: the relation between number of objective and the speedup

Again, it is clear from table 4 that the

new system does not only beat the existig system when comparing the execution time,but, also the quality of the result is better in all cases.

6 Conclusion

The GPU can increase the performance of computation with a good significant speedup, that we can use in many applications needing a fast computation.

The CUDA architecture differ from the other parallel architecture; it's designed to handle the mathematical and logical operations with less capability to handle other operations. So the performance of the systems runs on GPU can be increased if the programmer makes a good software design and makes the GPU busy all the time.

Since the CUDA only execute subprograms, the main controller will stay the CPU and that will cause transfers of data between the host (computer) memory and the device (GPU card) memory and vise versa. The memory access is time consumption and can't be ignored; so the designer of the system should try to minimize the memory transfer mainly when the problem does not take much time of execution. This problem could be shown clearly in our case when the dimensionality of the benchmark is 10.

In many implementations, when they try to work on performance enhancement, the developers do not highlight the quality of the system after improvement. In our implementation, we work on both, performance and quality. The result indicates that we got a better result than the existing system.

Many related implementation were built on the parallel MOGA that should work on many computers with CUDA enabled devices. By comparing the results with others, we can see that we got a general purpose tool that deliver a good performance on a single computer with better quality.

References

[1] NVIDIA Corporation. Nvidia cuda c programming best practices guide. 2009.

[2] NVIDIA Corporation. Nvidia cuda programming guide, version 2. 2009.

[3] E.P.Ephzibah. Cost effective approach on feature selection using genetic algorithms and ls-svm classifier. *IJCA Special Issue on Evolutionary Computation*, (1):16–20, 2010. Published by Foundation of Computer Science.

[4] R. Sophia Porchelvi i. An algorithmic approach to multi objective fuzzy linear programming problem international journal of algorithms. *International Journal of Algorithms, Computing and Mathematics*, 3:61–66, November 2010.

[5] Viryanet INC. The balancing act of mobile workforce planning fulfilling multiservice objectives with priority based optimization. *International Journal of Algorithms, Computing and Mathematics*, 2010.

[6] Sathish AP Kumar Jason P. Duran. Cuda based multi objective parallel genetic algorithms: Adapting evolutionary algorithms for document searches. *EEE'11 conferance*, July 2011.

[7] D. F. Jones, S. K. Mirrazavi, and M. Tamiz. Multi-objective metaheuristics: An overview of the current state-of-the-art. *European Journal of Operational Research*, 137(1):1–9, February 2002.

[8] Sarnath Kannan and Raghavendra Ganji. Porting autodock to cuda. In *IEEE Congress on Evolutionary Computation*, pages 1–8. IEEE, 2010.

[9] Nipen M Shah, Hubert Thieriot, Franois Marchal, and Andrew Hoadley. *A comparison of multi-objective optimisation approaches for a gas-phase refrigeration process*. 2008.

[10] Melanie Mitchell. *An introduction to genetic algorithms*. A Bradford book. MIT Press, Cambridge, Mass. [u.a.], 1996.

[11] Asim Munawar, Mohamed Wahib, Masaharu Munetomo, and Kiyoshi Akama. Hybrid of genetic algorithm and local search to solve max-sat problem using nvidia cuda framework. *Genetic Programming and Evolvable Machines*, 10:391–415, December 2009.

[12] Petr Pospchal and Josef Schwarz and Ji Jaro. Parallel genetic algorithm solving 0/1 knapsack problem running on the gpu. In *16th International Conference on Soft Computing MENDEL 2010*, pages 64–70. Brno University of Technology, 2010.

[13] M. Srinivas and Lalit M. Patnaik. Genetic algorithms: A survey. *IEEE Computer*, 27(6):17–26, 1994.

[14] Teimour Tajdari. Gpu implementation using cuda, master thesis report. 2009.

[15] Thomas Weise. *Global Optimization Algorithms – Theory and Application*. it-weise.de (self-published): Germany, 2009.

[16] Jialong Wu and Alice M. Agogino. Automating keyphrase extraction with multi-objective genetic algorithms. In *HICSS*, 2004.

[17] E. Zitzler, M. Laumanns, and S. Bleuler. A Tutorial on Evolutionary Multiobjective Optimization. In X. Gandibleux et al., editors, *Metaheuristics for Multiobjective Optimisation*, volume 535 of *Lecture Notes in Economics and Mathematical Systems*. Springer, 2004.