Implementation of GPU using Fine-Grained Parallel Genetic Algorithm

R.K.Nayak
School of Computer Engg.
KIIT University

B.S.P.Mishra
School of Computer Engg.
KIIT University

ABSTRACT
Many optimization problems have complex search space, which either increase the solving problem time or finish searching without obtaining the best solution. Genetic Algorithm (GA) is an optimization technique used in solving many practical problems in science, engineering, and business domains. Parallel Genetic Algorithm (PGA) has been widely used to increase speed of GA, especially after the spread of parallel platforms such as GPUs, FPGA, and Multi-Core Processors. In this paper, we introduce a type of PGA called Fine-grained Parallel Genetic Algorithm, which has the advantages of maintaining better population diversity, and inhibiting premature. Fine-grained PGA is implemented on graphics hardware.

Keywords
Parallel Genetic algorithm, FPGA, GPU, Parallel Processing

1. INTRODUCTION
The GA is one of the most important soft computing tools used to solve many engineering optimization problems. Being a soft computing tool, the evolution of a solution in GA is a non-deterministic process[1]. Hence, many times, while working on GA, one needs to deal with problem of convergence or premature convergence. Though the single population GA has excellent search performance but performance can be improved by increasing the population size or having more than one population [2]. As many GA solutions require a significant amount of computation time, a number of parallel genetic algorithms (PGAs) have been proposed in past decades [3] [4]. These algorithms differ principally from the classical sequential genetic algorithm, but they seem to have even better optimization quality [5]. For some kind of problems, the population needs to be very large and the memory required to store each individual may be considerable. In some cases this makes it impossible to run an application efficiently using a single machine, so some parallel form of GA is necessary. Fitness evaluation is usually very time-consuming. In the literature computation times of more than 1 CPU year have been reported for a single run in complex domains. It stands to reason that the only practical way of provide this CPU power is to the use of parallel processing. Sequential GAs may get trapped in a sub-optimal region of the search space thus becoming unable to find better quality solutions. PGAs can search in parallel different subspaces of the search space, thus making it less likely to become trapped by low-quality subspaces [6]. To tackle all these problems, mentioned above, one can use the GPGPU and can exploit its functionality to solve GA effectively with more speed up. Now a days there is a need of performance improvement in each application so as to minimize the time required for execution. The graphics processing unit (GPU) based computing is a broad area under which highly computational problems are solved. The GPU attracted many researchers because of its low-cost, high-performance computing and high availability. Though the GA's algorithmic development is at extreme level, but there is full scope for making it parallel on GPUs and its performance can be improved by minimizing data transfer between a CPU and a GPU by executing its operations of evaluation, selection, and reproduction through GPU [7].

2. BACKGROUNDS
2.1 Genetic Algorithms
Genetic Algorithms (GAs) are adaptive heuristic search algorithm based on the evolutionary ideas of natural selection and genetics. As such they represent an intelligent exploitation of a random search used to solve optimization problems. Although randomized, GAs are by no means random, instead they exploit historical information to direct the search into the region of better performance within the search space. The basic techniques of the GAs are designed to simulate processes in natural systems necessary for evolution, especially those follow the principles first laid down by Charles Darwin of "survival of the fittest."Since in nature, competition among individuals for scanty resources results in the fittest individuals dominating over the weaker ones. As described in Golberg [1]: in general terms, a genetic algorithm

consists of four parts.

1. Generate an initial population.

2. Select pair of individuals based on the fitness function.

3. Produce next generation from the selected pairs by applying pre-selected

genetic operators.

4. If the termination condition is satisfied stop, else go to step 2.

The termination condition can be either:

1. No improvement in the solution after a certain number of generation.

2. The solution converges to a pre-determined threshold.

GAs simulates the survival of the fittest among individuals over consecutive generation for solving a problem. Each generation consists of a population of character strings that are analogous to the chromosome. Each individual represents a point in a search space and a possible solution. The individuals in the population are then made to go through a process of evolution. GAs is based on an analogy with the genetic structure and behavior of chromosomes within a population of individuals using the following foundations:

- Individuals in a population compete for resources and mates.
- Those individuals most successful in each 'competition' will produce more offspring than those individuals that perform poorly.
- Genes from `good' individuals propagate throughout the population so that two good parents will sometimes produce offspring that are better than either parent.
- Thus each successive generation will become more suited to their environment.

Search Space

A population of individuals are maintained within search space for a GA, each representing a possible solution to a given problem. Each individual is coded as a finite length vector of components, or variables, in terms of some alphabet, usually the binary alphabet **{**0,1**}**. To continue the genetic analogy these individuals are likened to chromosomes and the variables are analogous to genes. Thus a chromosome (solution) is composed of several genes (variables). A fitness score is assigned to each solution representing the abilities of an individual to `compete'. The individual with the optimal (or generally near optimal) fitness score is sought. The GA aims to use selective `breeding' of the solutions to produce `offspring' better than the parents by combining information from the chromosomes. The GA maintains a population of n chromosomes (solutions) with associated fitness values. Parents are selected to mate, on the basis of their fitness, producing offspring via a reproductive plan. Consequently highly fit solutions are given more opportunities to reproduce, so that offspring inherit characteristics from each parent. As parents mate and produce offspring, room must be made for the new arrivals since the population is kept at a static size. Individuals in the population die and are replaced by the new solutions, eventually creating a new generation once all mating opportunities in the old population have been exhausted. In this way it is hoped that over successive generations better solutions will thrive while the least fit solutions die out. New generations of solutions are produced containing, on average, more good genes than a typical solution in a previous generation. Each successive generation will contain more good `partial solutions' than previous generations. Eventually, once the population has converged and is not producing offspring noticeably different from those in previous generations, the algorithm itself is said to have converged to a set of solutions to the problem at hand.

2.2 Parallel Genetic Algorithm(PGA)

It has long been noted that genetic algorithms are well suited for parallel execution. The different PGAs are optimized for different uses and implementations. Parallelization for the sake of being able to run the algorithm faster with the help of multiple processors. Keeping relatively isolated subpopulations is a parallelization method for acquiring greater diversity. PGAs can be divided into three general classes Master-Slave , Coarse-grained and Fine-Grained. In a master-slave model, there is a single population just as in sequential GA, but the evaluation of fitness is distributed among several processors. In a coarse-grained model, the GA population is divided into multiple subpopulations. Each subpopulation evolves independently, with only occasional exchanges of individuals between subpopulations. In a fine-grained model, individuals are commonly mapped onto a 2D lattice, with one individual per node. Selection and crossover are restricted to a small and overlapping neighborhood.

2.2.1 Master Slave Model

Master-slave Model (MSM) is mainly a variation to increase speed, scale and calculation power for GA. Distribution of crossover and mutation operations, and in some cases fitness calculation, can be done to different processors. This allows utilization of the computing power of several processors or distributed computer systems to solve the problem [4]. With the use of the additional resources from distributed computing, we are not as dependent on the development of hardware for single systems, to be able to be compute more complex problems. On single systems such problems might not be possible to solve within a reasonable time. MSM assigns a fraction of the population to the available processors for evolutionary operation. Then it can work in two ways. The first one is synchronous, which only has the benefit of faster computing. Once the population is assigned to the processors, synchronous MSM waits for all the processors to complete their operations and return the result, before evaluating the new population. The second one is asynchronous, which does not wait for slow processors to return their result. These work a bit different than standard GA.

2.2.2 Fine-grained Parallel Genetic Algorithm

Fine-grained Parallel Genetic Algorithm (FGA) [2] has only a single population. It has a special structure, which restricts the individuals in the population to only interact directly with it's neighbors. It has a problem that the performance of the algorithm degrades as the size of the population increases.

2.2.3 Coarse Grained Parallel Genetic Algorithm

Coarse-grained Parallel Genetic Algorithm (CGA) is also known as multiple-deme PGA or "island" PGA [2]. CGA divides the population into subpopulations which can be computed on separate processors [15]. The sub population then has an amount of migration between them. Benefits of having smaller populations are that favorable traits spread faster within the population. The drawback is that the rapid rise of fitness stops at a lower fitness value than with a single large population. At first sight this seems as a very simple method to use. The problems arise when implementations of it are analyzed more closely. When setting a low migration rate between the populations, we get the previously mentioned problem with a low quality of the overall solution. But if we should increase the migration level too much, the behavior changes back to that of a single large population [8].

2.3 Graphical Processing Unit (GPU

In the early 1990s, ubiquitous interactive 3D graphics was still the stuff of science fiction. By the end of the decade, nearly every new computer contained a graphics processing unit (GPU) dedicated to providing a high-performance, visually rich, interactive 3D experience [13][14]. This dramatic shift was the inevitable consequence of consumer demand for videogames, advances in manufacturing technology, and the exploitation of the inherent parallelism in the feed-forward graphics pipeline. Today, the raw computational power of a GPU was that of the most powerful CPU, and the gap is steadily widening. Furthermore, GPUs have moved away from the traditional fixed-function 3D graphics pipeline toward a flexible general-purpose computational engine. Today, GPUs can implement many parallel algorithms directly using graphics hardware. Well-suited algorithms that leverage all the underlying computational horsepower often achieve tremendous speedups. Truly, the GPU is the first widely deployed commodity desktop parallel computer. In the following diagram working of GPU is explained.

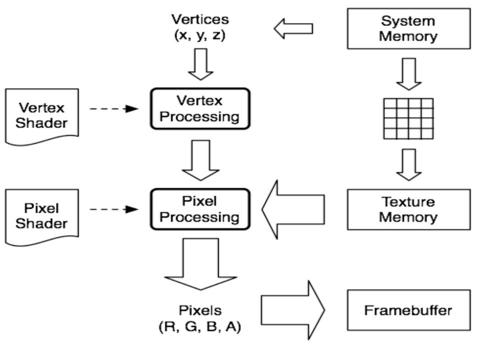

Fig1: Working of GPU

3. A REAL CODED PGA ON GPU

3.1 Algorithm Overview

In this paper, we adopt the fine-grained parallel model suitable for SIMD implementation. In this model each processor is allocated only one individual. That is to say that each subpopulation is composed of only one individual and it communicates with another one, with 1 Hamming distance between them [10]. It has advantages of maintaining better population diversity, inhibiting premature, keeping the utmost parallelism, and therefore outperforms all other traditional genetic algorithms when dealing with high-dimensional variable spaces [11].

```
for each node do in parallel
        generate an individual randomly
end parallel do
        while not stop_criterion_satisfied do
            for each node do in parallel
            evaluate the fitness of the individual
                get the fitness values of individuals
                select the individuals by selection rate
                crossover with the local
                individual according to the crossover rate
            mutate the individual according to the
            mutation rate
  end parallel do
                check the stopping criteria
        end while
```

3.2 Implementation of Genetic Operators

Selection, crossover and mutation operators described as:

Tournament selection

It is a method of selecting an individual from a population of individuals in a genetic algorithm [11,12]. Tournament selection involves running several "tournaments" among a few individuals chosen at random from the population. The winner of each tournament (the one with the best fitness) is selected for crossover. Selection pressure is easily adjusted by changing the tournament size. If the tournament size is larger, weak individuals have a smaller chance to be selected.

Tournament selection pseudo code:

choose k (the tournament size) individuals from the population at random

choose the best individual from pool/tournament with probability p
choose the second best individual with probability p*(1-p)
choose the third best individual with probability p*((1-p)^2)
and so on...

Arithmetic crossover

A crossover operator that linearly combines two parent chromosome vectors to produce two new offspring according to the following equations:

Offspring1 = a * *Parent1* + (1- a) * *Parent2*
Offspring2 = (1 – a) * *Parent1* + a * *Parent2*

where a is a random weighting factor (chosen before each crossover operation).

Consider the following 2 parents (each consisting of 4 float genes) which have been selected for crossover:

Parent 1: (0.3)(1.4)(0.2)(7.4)
Parent 2: (0.5)(4.5)(0.1)(5.6)

If a = 0.7, the following two offspring would be produced:

Offspring1: (0.36)(2.33)(0.17)(6.86)
Offspring2: (0.402)(2.981)(0.149)(6.842)

Gaussian Mutation

Let $x\epsilon[a,b]$be a real variable. Then the Gaussian mutation operator MG changes x to

MG:=min(max(N(x,σ),a),b),

where σ may depend on the length l :=b-a of the interval and typically σ/l=1/10. MG is applied with probability pm to each variable, where pm generally has a value of a few percent. The value of σ may also depend on time, i.e., the number of the current generation, and usually decreases with time. The reason for a decreasing σ is that stronger mutation during the beginning of an optimization supports the sampling of the search space and smaller displacements towards the end aid in fine tuning extreme values.

Another advantageous setup is to use two mutation operators: one with a high and one with a low value of σ. This enables to achieve a similar effect as a decreasing σ.

4. EXPERIMENTAL RESULTS

Our performance results were measured using an AMD Athlon 2500+ Intel Core i7-4930k@3.40GHZand an NVidia GeForce 6800GS GPU. The GPU-based implementation was developed with matlab code. The GA parameters applied as: crossover rate 1.0 and mutation rate 0.03, number of generations 600, n different population size. In the table GPU-based implementation was compared with software implementation running on single CPU with different population size. We see that speedup increases as we increase the population size. We used Chichinadze global optimization problem as benchmark. This is a multimodal minimization problem defined as follows:

$$F_{Chichinadze}(X)=x^2_1 – 12x_1 + 8 \sin(\tfrac{5}{2}\pi x_1) + 10 \cos\left(\tfrac{1}{2}\pi x_1\right)+ 11- 0.2\,\frac{\sqrt{5}}{e^{\frac{1}{2}}\,(x_2 - 0.5)^2}$$

Here, n represents the number of dimensions and $x_i \in [-30, 30]$ for i=1,2.

Population Size	**Genetic Operators**			**Fitness Evaluation**		
	GPU(s)	CPU(s)	Speedup	GPU(s)	CPU(s)	Speedup
34^2	0.211	0.296	1.4x	0.044	0.013	0.3x
68^2	0.262	1.201	5.8x	0.046	0.062	1.4x
136^2	0.444	5.230	11.8x	0.074	0.587	7.9x

Table 1. Time cost and speed up for different GA module (600 generations)

5. CONCLUSION

In this work, we have presented a novel implementation of parallel genetic algorithms on commodity graphics hardware. Our approach gives a representation of population suitable for GPU processing. All genetic operators have been implemented on GPU. Tests on a function optimization problem show that the larger the population size is, the better speedup over the software implementation can be achieved. Our work has provided a promising platform for implementation of PGAs. Looking toward future, programmable GPUs are on a much faster performance growth than CPUs. They also have many other advantages: inexpensive, readily available, easily upgradeable, and compatible with various operating systems and hardware architectures. There are still several constrains in our approach. For problems whose fitness function is not suitable for GPU implementation, the performance of our method will be seriously limited because of the bottleneck of transferring data between system memory and video memory in each GA loop. Another limitation is that commonly used binary encoding scheme of GAs seems hard to be implemented on the GPU because there is no bit-operator supported in current GPUs. In the future, we will apply the presented approach in real-world problems such as GA-based image processing. Another future work is further implementations of other variants of genetic algorithms.

6. REFERENCES

[1] Goldberg, D.E. 1989 .Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley Professional, First Edition.

[2] Umbarkar1, A.J. and Joshi2, M.S. and Rothe3, N.M. and Tomassini, M. 1995. A survey of parallel genetic algorithms. World Scientific III.

[3] Konfrst, Z. 2004. Parallel genetic algorithms: advances, computing rends, applications and perspectives. In Parallel and Distributed Processing Symposium.

[4] Wang, Chen,K. and Ong, Y.S. (Eds.).2005.Parallel Genetic Algorithms on Programmable Graphics Hardware. Springer-Verlag Berlin Heidelberg.

[5] M. Nowostawski, M. and Poli,R .1999. Parallel Genetic Algorithm Taxonomy. Vol. 13, MAY, 1999.

[6] Umbarkar, A.J. and Joshi, M.S. Joshi and Rothe, N.M. 2013. Genetic algorithm on general purpose graphics processing unit: parallelism review ictact journal on soft computing, january 2013, volume: 03, issue: 02

[7] Hassani,A. and Treijs,J.2009. An Overview of Standard and Parallel Genetic Algorithms.

[8] Al-marakeby, A.2013. FPGA on FPGA: Implementation of Fine-grained Parallel Genetic Algorithm on Field Programmable Gate Array,IJCA, vol-80, No 6, 2013.

[9] XUE Shengjun and GUO Shaoyong and BAI Dongling,2008. The Analysis and Research of Parallel Genetic Algorithm. Wireless Communications, Networking and Mobile Computing.

[10] Jian-Ming Li and Xiao-Jing Wang and Rong-Sheng He and Zhong-Xian Chi.2007. An Efficient Fine-grained Parallel Genetic Algorithm Based on GPU-Accelerated. International Conference on Network and Parallel Computing.

[11] Raghuwanshi, M., Kakde, O.: Survey on multiobjective evolutionary and real coded genetic algorithms. In: The 8th Asia Pacific Symposium on Intelligent and Evolutionary Systems, Cairns, Australia (2004)

[12] Michalewicz, Z.1996.Genetic Algorithms + Data Structures = Evolution Programs.

[13] Press, W.H.and Teukolsky, S.A.and Vetterling, W.T.2002. Flannery, B.P.: Numerical Recipes in C++. The Art of Scientific Computing. Cambridge University.

[14] Lukac, R.. and Plataniotis, K.N. and Smolka, B.2004. Venetsanopoulos, A.N.Color image filtering and enhancement based on genetic algorithms. IEEE International Symposium on Circuits and Systems.

[15] Houston, M.and Fatahalian, K. and Sugerman, J.and Buck, I. and Hanrahan, P.2004. Parallel computation on a cluster of gpus. ACM Workshop on General-Purpose Computing on Graphics Processors, Los Angeles,California.