*Reason - A Technical Journal*
ISSN 2277–1654
Volume - XIV ● 2015

# OPTIMIZATION OF TRUSS STRUCTURE USING GENETIC ALGORITHM PERFORMED ON GPU

**Subhajit Sanfui[1] and Ashish V. Gajbhiye[2]**

[1]Dept. of Mechanical Engineering, IIT Guwahati, Guwahati, India, Email: s.sanfui@iitg.ernet.in
[2]Dept. of Mechanical Engineering, IIT Guwahati, Guwahati, India, Email: a.gajbhiye@iitg.ernet.in

**Paper received on: December 23, 2014, accepted after revision on: November 02, 2015**

**Abstract:** Modern Graphics Processing Units (GPU), offer a tremendous computing power, that is frequently an order of magnitude larger than even the most modern multi-core CPUs, making them an attractive platform for high performance computing due to their relative cheapness compared with conventional PC clusters. General purpose computing on GPUs (GPGPU) is becoming popular in High Performance Computing (HPC) because of its high peak performance. In this paper, a typical two-dimensional truss structure optimization problem is solved using Binary Genetic Algorithm (BGA) on both CPU and GPU. The kernel inside the GPU code computes the nodal displacements and elemental stresses by Finite Element Analysis (FEA) to evaluate the objective function and the constraints while making use of the Single Instruction Multiple Data (SIMD) structure of GPU to attain parallelization. The results are assessed for different values of parameters, such as complexity of the problem, number of elements, population size, maximum allowable generations and number of threads etc. to demonstrate how the value of speedup varies with these parameters and to provide a basic guideline for choosing the parameters for a different problem. The results clearly establish that calculations are performed considerably faster through the GPU than through the CPU in general.

**Keywords:** GPU; Genetic Algorithm; GPGPU; Topology Optimization.

## 1. Introduction

Modern computational devices are becoming more and more parallel, while GPUs are at the leading edge of this trend. The development of programmable GPUs opens up a new area of research, enabling the use of them for processing non-graphic compute-intensive problems. The enormous computational power available in modern GPUs can pave the way for their usage in HPC. The peak performance of a modern CPU is approximately 70 GFlops, while the latest high-end GPUs boast over 4.29 TFlops of performance. This enormous difference in computing capability has driven a multitude of modern researchers from several fields to exploit the inherent parallelism in several computationally intensive problems for attaining performance [1].

The simple truss optimization problem, used in this paper has already been solved by a number of researchers, as the stepping stone of structural optimization. Evolutionary computational techniques are probably one of the best ways to solve this kind of problem, as shown by many researchers in this field. It has also been shown that among all the evolutionary techniques available, GA often provides superior results, as compared to other other complicated and specialized methods [2, 3]. Goldberg and Samtani [4] appear to have first suggested the use of GAs for structural optimization. They considered the use of a GA to optimize a 10-bar plane truss. A few others have applied the technique to the design of welded beams [5], plane frames [6], a trussed-beam roof structure and a thin-walled cross section, and generalized trusses [7-9]. Among the more recent works, Ruiyi et al. [10] solved the truss topology optimization problem using GA with an Individual Identification Technique to get rid of redundant and repetitive computation. In 2014, Cazacu et al. [9] applied GA for optimization of steel trusses on MATLAB using a specialized penalty function. Conforming to this trend of using GA for truss optimization problems, this paper primarily aims at further reducing the computation time.

The programming language used for coding on the CPU and the GPU are C/C++ and CUDA respectively. CUDA is the parallel computing platform and programming model created by NVIDIA. The GPU used in the present study is NVidia Tesla C2075, with 448 CUDA cores and 6 Gigabytes of global memory. A number of different structures are optimized using the same algorithm in this work to establish the efficiency of the GPU. The primary focus is kept on the comparison of the same results, computed on the CPU and the GPU. In the next section a detailed statement of the truss design problem, and in the following sections, details of the optimization algorithm are presented. These are followed by the obtained results and conclusions.

## 2. PROBLEM DESCRIPTION

### 2.1 Basic Problem Statement

Optimization of the topology of a two dimensional truss structure (Fig. 1) is aimed at by minimizing its weight and/or the maximum stress/deflection occurring at any node, subjected to the constraints that values of the nodal deflections and values of the elemental stresses should not exceed certain permissible values.

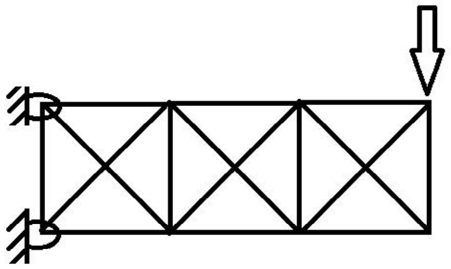

**Fig.1.** A typical truss structure

### 2.2 Mathematical Formulation

Mathematical formulation to solve the present problem is made as follows:

1. Maximize F = 1/(Weight of the structure)

Subject to $\sigma_i <= \sigma_{yield}$ for i=1,2,…,16

$\Delta_i <= \Delta_{max}$ i=1,2,…, 8

2. Maximize F = $1/((w_1*\text{weight})+(w_2*\Delta_{max}))$

Subject to $\sigma_i <= \sigma_{yield}$ for i=1, 2,…, 16

$\Delta_i <= \Delta_{max}$ i=1,2,…, 8

The first problem is a single objective optimization problem whereas the second one is a multi-objective one. The weights $w_1$ and $w_2$ are set through trial and error. $\sigma_i$ and "$_i$ denote respectively the stress in the ith member and deflection of the ith node. $\sigma_{yield}$ and $\Delta_{max}$ represent the maximum permissible values of the stresses and deflection. The objective function is taken as a maximization one because the algorithm uses proportionate selection method, which is suitable for maximization problem in general.

## 3. OPTIMIZATION METHOD AND ALGORITHM

### 3.1 Optimization Method

Binary Genetic Algorithm (BGA) is selected as the optimization method to solve the problem discussed in the previous section. Gas differ from traditional optimization methods in many ways [3]. The reasons of choosing BGA are as follows:

1. Works with a population of solutions instead of one. Hence, a number of solutions with equal or close objective function value can be obtained.
2. Does not require problem specific knowledge to carry out a search.
3. A particular configuration of the structure can be represented by a binary string very easily.
4. Evolutionary algorithms tend to be efficient in solving structural optimization problems in general.
5. Because of the stochastic nature of GA, it appears to be robust in noisy environments.
6. GAs operate on multiple partial solutions simultaneously (sometimes called implicit parallelism), gathering information from a population of search points to direct subsequent search efforts. Their ability to

maintain multiple partial solutions concurrently helps make GAs less susceptible to the problems of local maxima and noise [2].

Details of the BGA used in this paper are as follows:

1. Proportionate Selection
2. Single Point Crossover
3. Bitwise Mutation
4. (μ, λ) / (μ + λ) Survival Scheme
5. Population 100
6. Crossover Probability 0.9
7. Mutation Probability 0.25

The values of the crossover and mutation probability are set through trial and error to facilitate good convergence.

Fig.2 shows the typical flow of a BGA.

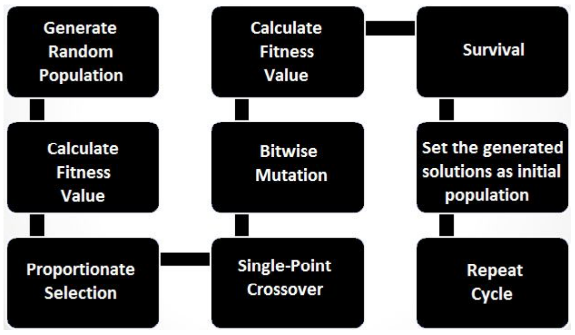

**Fig. 2.** General form of Genetic Algorithm

### 3.2 Details of the Algorithm

Fig. 3 and Fig. 4 represent the numbering sequence of the elements and the nodes respectively.

#### Solution Representation

According to the rule shown by the numberings in Fig. 3, any configuration of the truss can be represented by a 16 bit binary string.

e.g. 0011010110101101

#### Algorithm on CPU

On the CPU, the code (written in C) is executed as a serial one. The flow of the algorithm is as shown in Fig. 2.

#### Algorithm on GPU

On the GPU, the entire code may be divided into two separate parts,

a. Parallel Part

b. Serial Part

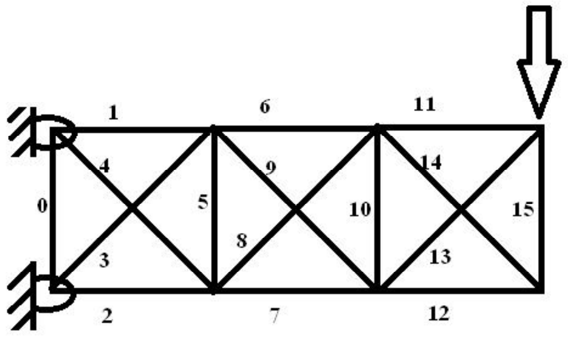

**Fig. 3**. Numbering of elements

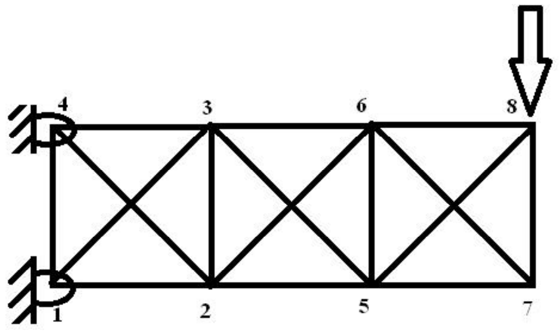

**Fig. 4.** Numbering of Nodes

For one run of the algorithm, the fitness of a number of individuals is calculated twice. First, inside the proportionate selection function (n individuals) and second, inside the survival function (2 *n individuals). The calculation of the fitness values of the individuals is carried out inside kernels, and hence is the parallel part of the code, whereas the rest of the code is carried out on the CPU, and hence, is the serial part of the code. The GPU and the CPU versions of the algorithm differ only in terms of calculating the fitness value. Fig. 5 and Fig. 6 show schematics of the calculation of fitness value on GPU and CPU respectively.

As shown in Fig. 5, the fitness values of all the individuals are carried out in parallel, and the values are stored in a one dimensional array on the device. Following this, the data is transferred back to the host (CPU) using the cudaMemcpy function of the standard CUDA library for further computation.

Two schemes of parallelism are implemented in this paper.

a. One Block, n Threads

b. n Blocks, 1 Thread

Results obtained from these two schemes are compared in the results section.

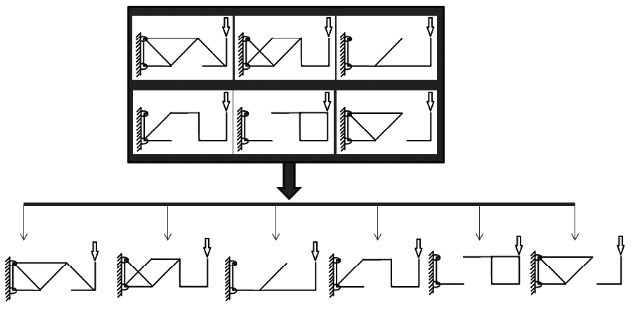

**Fig. 5.** Calculation of fitness inside GPU

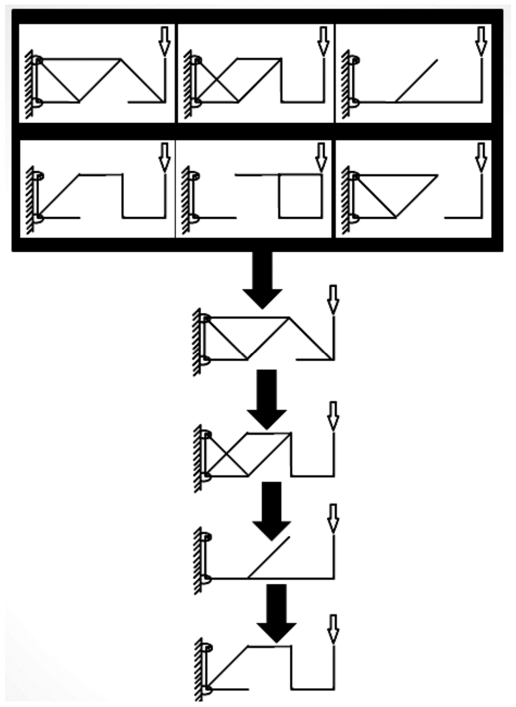

**Fig. 6.** Calculation of fitness inside CPU

Although, in this paper, parallelism is implemented only at the level of the optimization algorithm, by exploiting the natural parallelization features of GA, it can also be implemented at the level of structural analysis where complex finite element calculations are performed [8]. Following are the steps to take to calculate fitness.

1. To generate elemental stiffness for each member.
2. If member is present, E=Eactual.

If not present, E=Eactualx10-6

3. To generate global stiffness by assembling
4. To solve 16x16 system to find nodal deflections
5. To find stresses in each member
6. If Stress in each member is under yield stress, return (1/weight)
7. Else return (1/ (weight + penalty))

### 3.3 Modifications

To deal with infeasible solutions, following continuity conditions are imposed for directing the search towards feasible regions of the search space.

i. At least one of the elements from each of the the sets {0, 1, 2, 3, 4}, {6, 7, 8, 9} and {11, 13, 15} has to be present.

ii. Nodes 1, 4 & 8 have to be present.

iii. A hanging member is either removed, or another member is added such that it is not hanging anymore, based on a probability of 0.5.

## 4 RESULTS AND DISCUSSION

In this section, the resulting structures of the BGA for different yield stresses (High, Mediumand Low) and different objective functions (Problem 1 & 2) are presented, followed by the variation of the speedup with different parameters.For High and low values of the yield stress, the properties of Aramid (3600 MPa) and Cast Iron (130 MPa) are used respectively. For a Mid-Range yield stress, an intermediate value (500 MPa) is chosen.

### 4.1 Resulting Structures

Following resulting structures are obtained as the output of the algorithm.

- With yield stress set to a high value (Fig. 7)
- With yield stress set to a medium value (Fig. 8)
- With yield stress set to a low value (Fig. 9)
- Further lowering the yield stress (Fig. 10)
- With the multi-objective problem (Problem 2) (Fig. 11)

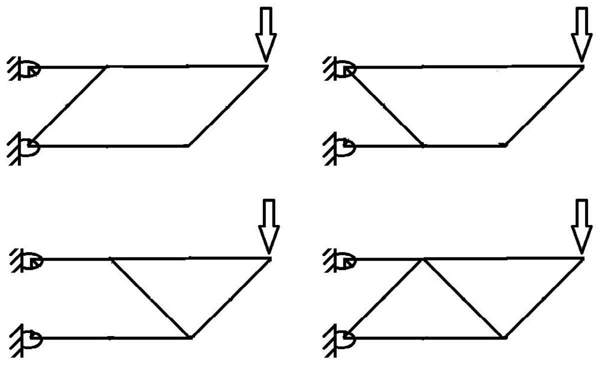

**Fig. 7.** Solutions with high yield stress

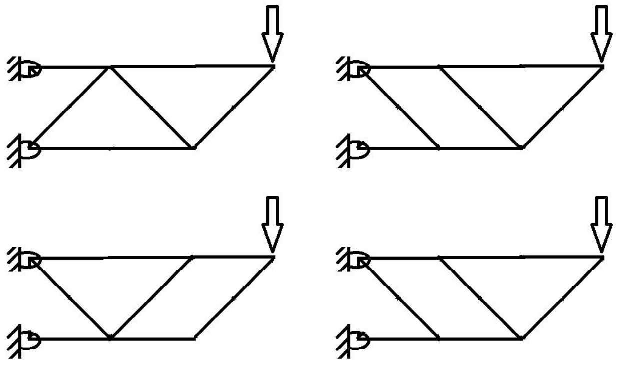

**Fig. 8**. Solutions with medium yield stress

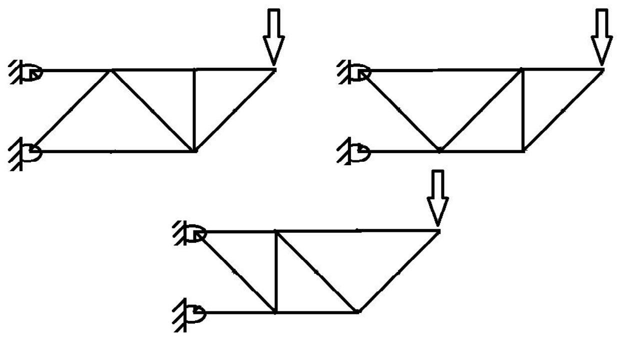

**Fig. 9**. Solutions with low yield stress

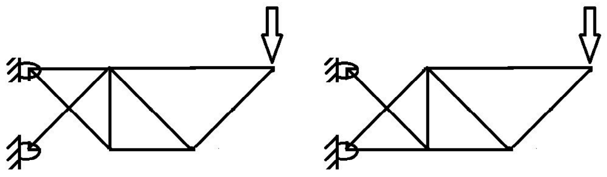

**Fig. 10.** Further lowering yield stress

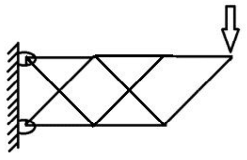

**Fig. 11.** Solution of problem 2

### 4.2 Variation of Speedup with Different Parameters

Fig. 12 shows the variation of run time with the number of population (same as number of threads) for a ‘One block multiple threads’ scheme.

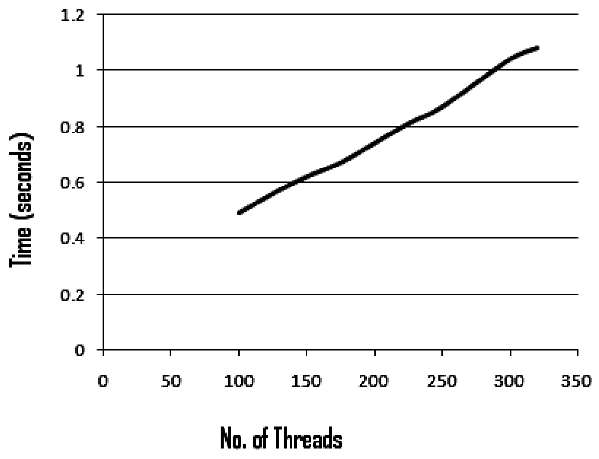

**Fig. 12**. Plot of time with variation of number of threads

Since the number of threads is set equal to the population size, with increase in number of threads, calculation time is also increased. The maximum number of threads usable for one block is 640 (obtained experimentally for the particular GPU in use). Since in the ($\mu + \lambda$) Survival Scheme, 2*n number of threads are to be utilized, the maximum permissible value of population becomes half of 640 or 320 in this parallelization scheme.

**Fig. 13** shows the variation of run time with the number of population (same as number of threads) for a 'Multiple blocks one thread' scheme.

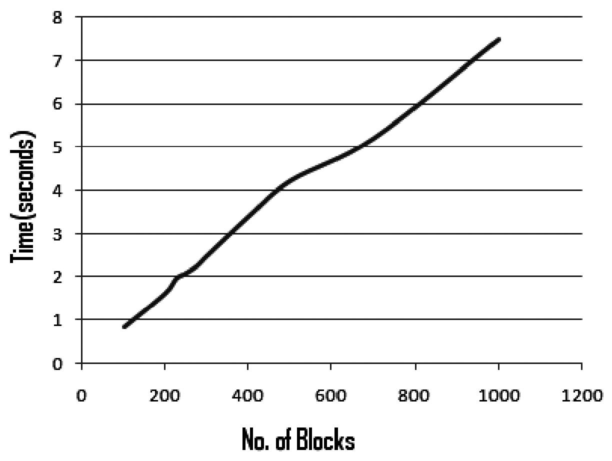

**Fig. 13.** Plot of time with variation of number of blocks

In this case also, since the number of blocks is set equal to the population size, calculation time increases linearly with increase in population size. However, in this scheme of parallelization, the limit on the number of blocks (and hence on the population size) is far greater than in the previous scheme.

Fig.14 shows the comparison between 'single thread multiple blocks' and 'single block multiple threads' schemes together.

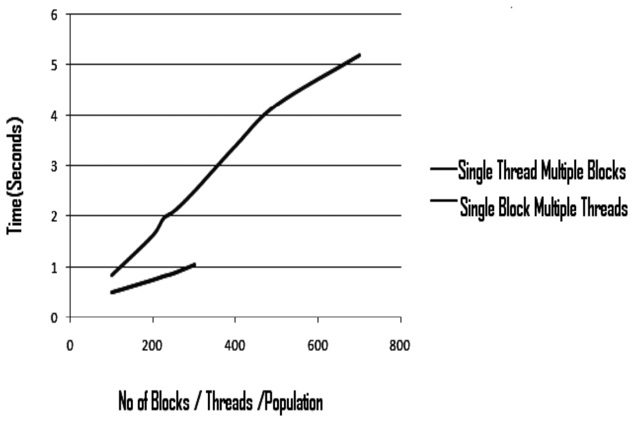

**Fig. 14.** Comparison of two schemes

This figure more than clearly indicates that the 'Single Block Multiple Threads' scheme is considerably more efficient than the 'Multiple Blocks Single Thread' scheme. It also indicates the limit for the multiple threads scheme, which is not present in the multiple blocks scheme.

The execution time on the GPU is calculated using the NVIDIA Profiler *nvprof*, whereas, that of the CPU is calculated using the *clock()* function. For a more accurate measurement of the execution time, more precise methods may be applied.For a population size of 100, the speedup obtained is 3.06, whereas, for a population size of 300, the attained speedup is 4.36.

Fig. 15 shows the variation of speedup with population size.

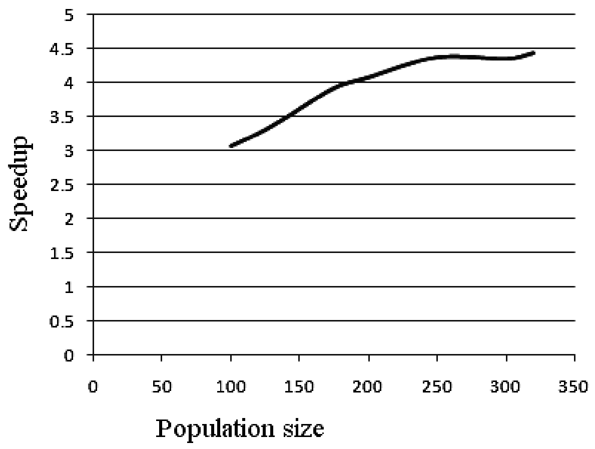

**Fig. 15.** Plot of Speedup with variation of Population Size

It can be clearly seen from the Fig. 15 that the speedup increases with increase in population size. However, the rate of increase decreases gradually. Beyond 250 population size, the curve becomes almost horizontal, indicating the invariance of speedup with population size.

Fig. 16 shows the comparison of Run time with the population size for both CPU and GPU.

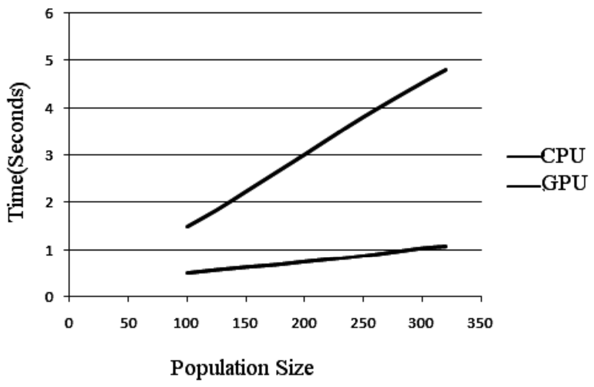

**Fig. 16**. Performance of GPU and CPU with population size

The slope for the CPU curve is higher than that of the GPU curve. For the same population size, computation time is considerably higher in CPU than in GPU. This difference becomes more pronounced with increase in population size.

## 4.3 Variation of Speedup vs Complexity of the Problem

To analyze the effect of increasing complexity of the problem on the speedup obtained, the algorithm is tested on two more problems of higher complexity. The two structures are shown in Fig. 17. Although 'complex' is a relative term, the number of nodes or number of elements may be taken as quantitative measurements of the complexity of the structural optimization problem. Compared to the number of elements, the number of nodes can serve as a better attribute for complexity, since it is directly related to the dimension of the linear system of equations to be solved in the Finite Element Analysis, which is twice the number of nodes in the structure.

The structures in Fig. 17 comprise of 29 elements, 12 nodes and 42 elements, 16 nodes respectively as compared to 16 elements and 8 nodes of the primary

problem. Optimization of this kind of structures requires complex continuity conditions to handle the problems of infeasible structure and hanging members. This paper does not aim at solving these problems efficiently i.e. handling the problems discussed in section III C. Instead it aims at establishing a basic relationship between the speedup and the increasing complexity of a problem.

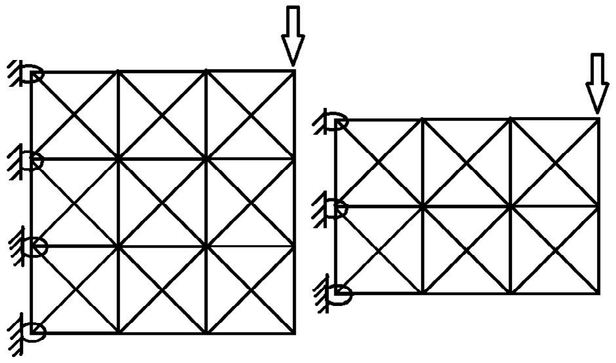

**Fig. 17.** Structures with 16 and 12 nodes

In a nutshell, the number of nodes and elements is taken as a measure of complexity of a structural optimization problem and is varied with the obtained speedups. Before going into the complexity versus speedup variation, the results for these problems are presented.

For the problem shown in Fig. 17, solutions are obtained as shown in Fig. 18 and Fig. 19.

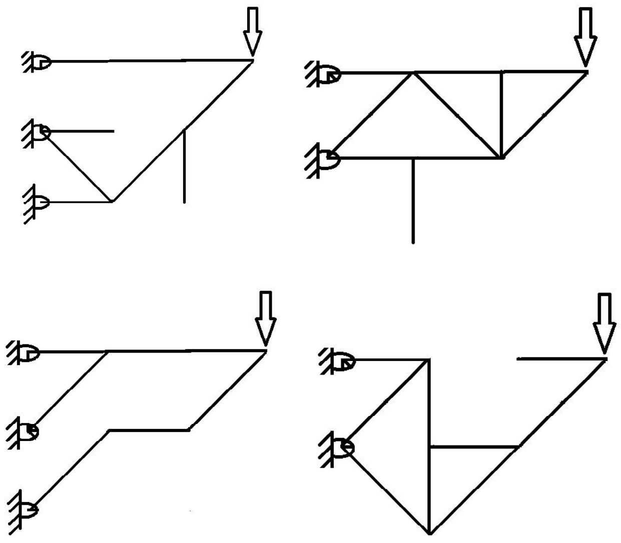

**Fig. 18**. Solutions of 29 element truss

As can be seen, while some of the results obtained are good, some do contain infeasibility and hanging members. Correction of these is out of scope of this paper. Comparison of speedups with the number of nodes and elements is shown in Fig. 20 and Fig. 21. From these two figures, it is clear that the speedup is more if complexity is high. The slope is higher in case of Fig. 21 than in case of 20.

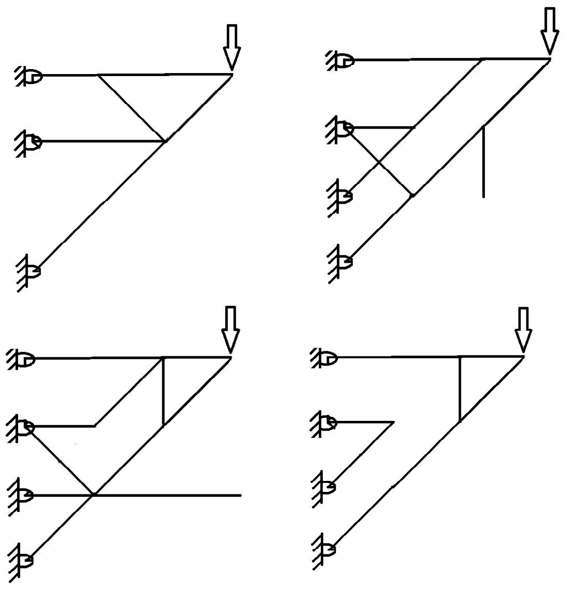

**Fig. 19.** Solutions of 42 element truss

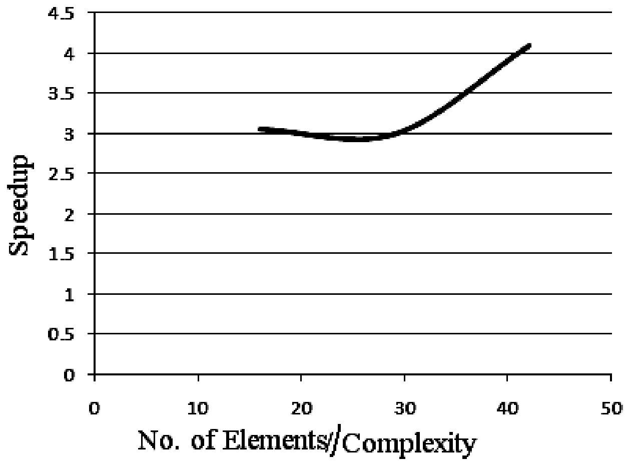

**Fig. 20.** Plot of Speedup with variation of Number of elements

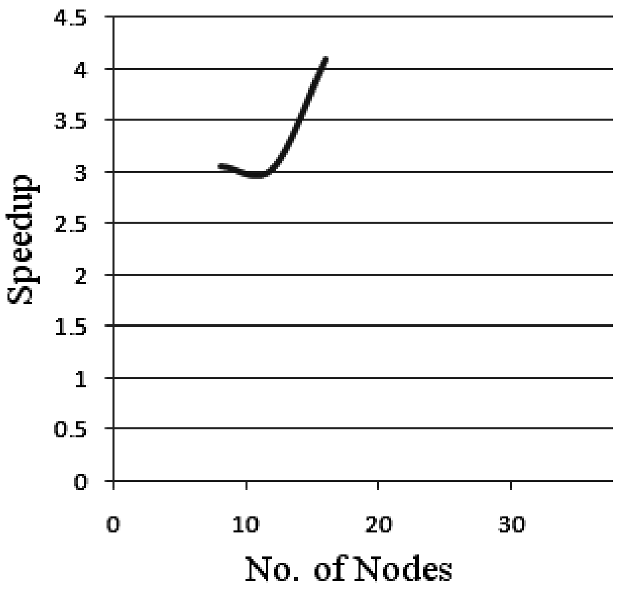

**Fig. 21.** Plot of Speedup with variation of Number of Nodes

Lastly, Fig. 22 shows the variation of speedup with the number of maximum generations of BGA. Here also, the speedup increases if the maximum number of generations in BGA is set to a higher value.

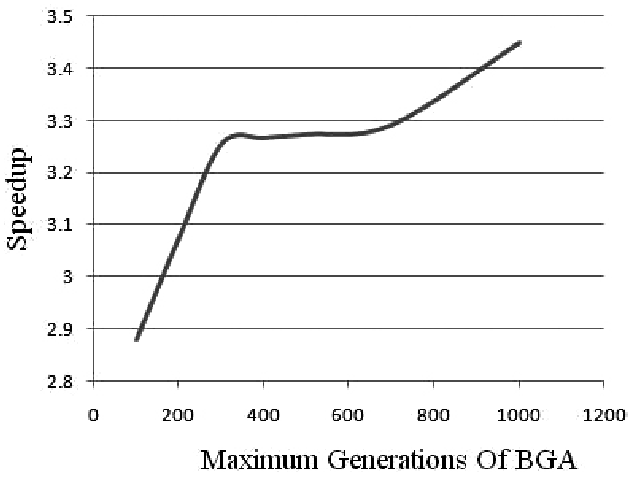

**Fig. 22.** Plot of Speedup with variation of Maximum generations

## 5 CONCLUSION

Based on the results shown in Section IV, it can be concluded that, for the same amount of computation, GPU takes considerably less time than the CPU in case of topology optimization problem using GA. The speedup increases with increasing population size, maximum generations of GA, complexity of the structure, complexity of the objective function, number of nodes and number of elements. Although GPU seems to be a lucrative choice for about any kind of problem, If only the complexity of the problem is above a certain limit, the problem is feasible for parallel implementation given the extra effort for coding and implementation. By comparison of the two schemes of parallelization (Single Block Multiple Threads & Multiple Block, Single threads), it is clear that Multiple threads scheme is faster, but it suffers a drawback of low maximum limit (640). For most of the conventional problems this is enough as the population size. But for certain problems a higher population size may be required, in which case, an alternative 'Multiple Blocks, Multiple Threads', schememay be employed.

**Acknowledgment:** Authors would like to thank Dr. Deepak Sharma for his contributions in this paper that helped in completion of this project.

## REFERENCES

[1] Chen, G., Li, G., Pei, S. and Wu, B. High Performance Computing Via a GPU, Proceedings of the 1st International Conference on Information Science and Engineering, 2009.

[2] Coello, C.A., Rudnick, M. and Christiansen, A.D., Using Genetic Algorithms for Optimal Design of Trusses, IEEE Event, Tulane University, New Orleans, 1994.

[3] Buckles, B.P. and Petry, F.E., Genetic Algorithms. Technology Series. IEEE Computer Society Press, 1992.

[4] Goldberg, D.E. and Samtani, M.P., Engineering Optimization Via Genetic Algorithm, Proceedings of Ninth Conference on Electronic Computation, New York, pp.471-482, 1986.

[5] Deb, K., Optimal Design of a Welded Beam via Genetic Algorithms, Proceedings of AIAA Journal, Vol.29, pp.2013-2015, 1991.

[6] Jenkins, W.M., Plane Frame Optimum Design Environment Based on Genetic Algorithm, Journal of Structural Engineering, Vol.118, No.11, pp.3103-3013, 1992.

[7] Rajeev, S., Krishnamoorthy, C.S., Discrete Optimization of Structures using Genetic Algorithms, Journal of Structural Engineering, Vol.118, No.5, pp.1233-1250, 1992.

[8] Papadrakakis, M., Lagaros, N.D. and Fragakis, Y., Parallel Computational Strategies for Structural Optimization, International Journal of Numerical Methods in Engng, 2003.

[9] Cazacu, R. and Grama, L., Steel Truss Optimization Using Genetic Algorithms and FEA, Procedia Technology, Vol.12, pp.339-346, 2014.

[10] Ruiyi, S., Liangjin, G. and Zijie, F., Truss Topology Optimization Using Genetic Algorithm with Individual Identification, Proceedings of the World Congress on Engineering, Vol. 2, 2009.