SIMP-based Structural Topology Optimization using Unstructured Mesh on GPU

Shashi Kant Ratnakar[1], Subhajit Sanfui[2] and Deepak Sharma[3]

Department of Mechanical Engineering, Indian Institute of Technology Guwahati, Assam – 781039, India
{r.shashi[1], s.sanfui[2], dsharma[3]}@iitg.ac.in

Abstract. Structural topology optimization is a method of finding optimum material distribution within a given design domain, which is subjected to various loading and support conditions. However, the large computation time is one of the major challenges in its implementation. This challenge gets escalate with the use of unstructured mesh. In this paper, a Solid Isotropic Material with Penalization (SIMP)-based implementation of topology optimization on the graphics processing unit (GPU) for a 3D unstructured mesh is presented. The finite element analysis is performed entirely on a GPU. The main implementational challenges are addressed by developing an efficient and optimized GPU $kernel$. The performance of the implementation is analyzed over an example using three different mesh sizes. Results show almost 4 × speedup over a standard CPU implementation.

Keywords: Topology Optimization, SIMP, GPU, Unstructured Mesh

1 Introduction

Topology optimization is a method for designing lightweight and reliable structures. Its main objective is to find an optimum layout by distributing material in a given design space. The material is distributed in order to achieve the best structural performance in terms of minimizing compliance, cost, or weight of the structure. The topology optimization has been successfully used in mechanical structural design [1], civil structures [2], aerospace structures [3], to name a few.

The theoretical aspect of topology optimization is well established. The density-based methods are popular and widely used, particularly SIMP [4]. The density-based methods work on a fixed domain of finite elements and represent a smooth and differentiable problem, which can be solved efficiently by available numerical optimization algorithms.

The SIMP -based topology optimization method begins with the discretization of a design domain, which is known as meshing. In literature, the structured mesh has been mostly used for meshing solid structures [5]. Structured meshes indeed offer a clear advantage in terms of simplicity of computational implementation, and the amount of computation required. However, their use is limited to simple design problems with

regular domain geometry, thereby making their limit applications for real-world problems.

The unstructured meshes are more versatile to discretize domains with complex and irregular geometries. This makes them useful over a wider range of problems, such as those containing non-orthogonal domains and curved boundaries. However, the main challenge is a higher computation time than that with a structured mesh. As topology optimization is already a computationally expensive method, its implementation using unstructured meshes makes it even more computationally challenging.

A viable solution to a high computation time can be found by performing the computation in parallel. The GPU has been very popular in the recent few years for solving data-parallel computation.

Previous studies used GPU to speedup different parts of topology optimization. The majority of these studies targeted the finite element analysis (FEA) part of the SIMP-based topology optimization [6, 7, 8, 9]. Schmidt et al. used GPU to accelerate FEA and gradient computation on a 3D structured mesh. Special attention was given for implementing a sparse matrix-vector product (SpMV) on GPU [5]. Several studies in the literature also computed both FEA and optimization on GPU [10, 11, 12]. Methods other than SIMP have also been used with GPU. For example, the level-set method [13], topological sensitivity method [9], and evolutionary optimization [14,15] have also been implemented on GPU. Recently, Kiran et al. presented a GPU-based strategy for the generation of finite element stiffness matrices and their assembly [16]. Sanfui et al. used GPU to accelerate the elemental computation and assembly by exploiting the symmetry [17]. Kiran et al. presented a comparative analysis of the GPU-based solver libraries for a sparse linear system of equations [18].

Most of the studies discussed above use structured meshes for discretization and analysis. The first work on GPU-based topology optimization using 2D unstructured meshes was presented by Zegard et al. [10]. Later, Duarte et al. presented a GPU-based topology optimization tool PolyTop++ [7] using a polygonal mesh. The following are the contributions of this paper.

- Implementation of SIMP-based topology optimization using 3D unstructured mesh on GPU. The implementation is developed using compute unified device architecture (CUDA) [19], to be run on NVIDIA GPUs.
- Development of efficient CUDA *kernel* for matrix-free SpMV and application of *Thrust* library [20] of the CUDA toolkit for linear algebra operations.
- Development of GPU strategy in order to perform FEA entirely on GPU, which thus obviates the need for CPU-GPU data transfer.
- Comparative analysis of the proposed GPU implementation with a standard CPU implementation.

The paper is organized as follows. The implementation details are discussed in Section 2. In Section 3, the results and discussions are presented, followed by the conclusions in Section 4.

2 Computational Implementation

The structural topology optimization for compliance minimization under volumetric constraint can be defined as,

$$\min_{\rho} \quad C(\rho) = \boldsymbol{u}^T K \boldsymbol{u}, \qquad Sub.to \quad K(\rho)\boldsymbol{u} = \boldsymbol{f}, V(\rho) = \sum_e \boldsymbol{\rho_e} \leq V^*, \boldsymbol{\rho}_e \in \{0,1\}, \tag{1}$$

where K is the stiffness matrix, $\boldsymbol{u}$ is the nodal displacement vector, $\boldsymbol{\rho_e}$ is the elemental density and V^* is the volume fraction for restricting material consumption.

The flowchart in Fig. 1 shows the computational steps involved in the proposed implementation of the topology optimization for compliance minimization of solid continua.

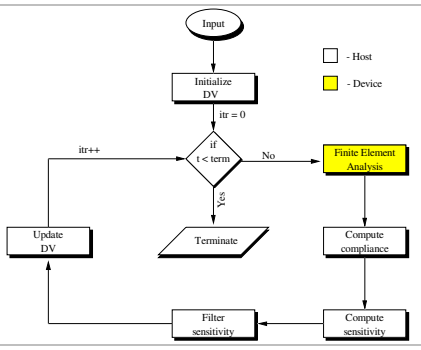


Fig. 1. Flowchart of topology optimization implementation

The input to the implementation includes the material properties, mesh information, and the code parameters (termination criterion and maximum no. of iterations). In the beginning, the decision variable (DV) vector is initialized. Since the proposed implementation uses SIMP -based topology optimization, the elemental densities are the decision variables. Thereafter, FEA is carried out to compute the nodal displacements. The profiling of the CPU implementation reveals that 99.5% of total computation time is being consumed by FEA, which makes it the most time-consuming block in the entire optimization shown in Fig. 1. Hence, in order to reduce the total computation time, the entire FEA process is carried out on GPU without the need for any CPU-GPU data transfer.

Once the nodal displacements($\boldsymbol{u}$) are known, the compliance of the structure is computed as per Eq. (2), and its sensitivity as per Eq. (3).

$$c(\boldsymbol{\rho}) = \sum_{e=1}^{N_{ele}} (\boldsymbol{\rho}_e)^p \{\boldsymbol{u}_e\}^T [k_e] \{\boldsymbol{u}_e\}. \tag{2}$$

$$\frac{\partial c}{\partial \boldsymbol{\rho}_e} = -p\, \boldsymbol{\rho}_e{}^{p-1} \{\boldsymbol{u}_e\}^T [k_e] \{\boldsymbol{u}_e\}. \tag{3}$$

where p is the penalty parameter, and N_{ele} is the total no. of elements in the mesh, $\boldsymbol{\rho_e}$ is the elemental density, $[k_e]$ is elemental stiffness matrix, and $\{\boldsymbol{u}_e\}$ is the vector of nodal displacements of the element e

The next step involves the mesh independency filter, which is an important step to prevent the checker-board problem in the resulting topologies [10]. Following this, the decision variable (density) vector is updated. The key details of the FEA procedure on

GPU is discussed in subsection 2.2. Since the theory of SIMP-based topology optimization is well established in the literature, the reader may refer to [4, 21] for further details.

2.1 Preliminary Computations

The preliminary computation includes reading the mesh information obtained from ANSYS and computing elemental stiffness matrices. The mesh information taken from ANSYS includes the nodal coordinates and the connectivity matrix. For all the elements $[K_e]$ is computed only once on the CPU. All three set of data is linearized following row-major order and copied on to the GPU global memory for further computation.

2.2 Finite Element Analysis on GPU

Once all the required data is transferred to the GPU, nodal displacements are computed using FEA in each iteration of topology optimization. This typically involves assembling the elemental stiffness matrices into the global stiffness matrix $[K]$ and the load vector $\{\boldsymbol{f}\}$**.** However, in the present study, a matrix-free approach is used, where [K] is not explicitly assembled. The next step is to solve the system of linear equations. A matrix-free conjugate-gradient (CG) method is used for the same. Algo. 1 shows the steps of the CG solver.

From the algorithm, it can be seen that each CG iteration involves one SpMV and multiple vector-vector multiplications. As discussed earlier, these SpMV operations are the main computational bottleneck in the whole topology optimization. In the present work, the entire CG routine is executed on GPU. For matrix-vector multiplications, custom CUDA kernels are developed. The rest of the linear algebraic operations are performed using the $Thrust$ library of CUDA toolkit, as shown in Algorithm 1.

First, the required data is copied into GPU. The data includes $[K_e]$ of all the elements, connectivity matrix $[C]$, and elemental densities $(\boldsymbol{\rho})$. Since, in the present implementation, each mesh is taken unstructured, the connectivity information is explicitly stored. The elemental densities are updated in every iteration, and the updated densities are sent to GPU. All the data are copied into the global memory of GPU.

SpMV Kernel: The SpMV is performed on GPU using an element-by-element (EbE) strategy. In this approach, $[K_e]$ of each element is multiplied with their respective $\{u\}$ components, and by performing this operation for all elements in the mesh, the final product $[K]\{u\}$ is achieved. This can be represented as

$$u_{new} = Ku = \sum_{e=1}^{N_{ele}} K_e u_e \,. \tag{4}$$

In the kernel design, each element of FE mesh is allocated to one compute $thread$ of GPU. This thread reads the connectivity information for the nodes of this element and the elemental density and performs SpMV. The pseudo-code of the SpMV kernel is given in Algo. 2.

Algorithm 1. Conjugate gradient algorithm

Data: $\boldsymbol{f}, K, \boldsymbol{u_0}, i_{max}, \varepsilon$
Output: $\boldsymbol{u}$

1. $i \leftarrow 0$
2. $\boldsymbol{r_0} \leftarrow \boldsymbol{f} - K\boldsymbol{u_0}$ /* Kernel */
3. $\boldsymbol{d_0} \leftarrow \boldsymbol{r_0}$ /* Thrust */
4. $\delta_{new} \leftarrow \boldsymbol{r_0^T r_0}$ /* Thrust */
5. $\delta_0 \leftarrow \delta_{new}$ /* Thrust */
6. **while** i < i_{max} **and** $\delta_{new} > \varepsilon^2 \delta_0$ **do**
7. $\boldsymbol{q} \leftarrow K\boldsymbol{d}$ /* Kernel */
8. $\boldsymbol{\alpha} \leftarrow \delta_{new} / \boldsymbol{d^T q}$
9. $\boldsymbol{u} \leftarrow \boldsymbol{u} + \alpha \boldsymbol{d}$ /* Thrust */
10. $\mathbf{r} \leftarrow \mathbf{r} - \alpha \mathbf{q}$ /* Thrust */
11. $\delta_{old} \leftarrow \delta_{new}$ /* Thrust */
12. $\delta_{new} \leftarrow \boldsymbol{r^T r}$ /* Thrust */
13. $\beta \leftarrow \delta_{new} / \delta_{old}$
14. $d \leftarrow r + \beta d$ /* Thrust */
15. $i \leftarrow i + 1$
16. **end**

Algorithm 2. SpMV kernel

Data: $K_e, \boldsymbol{\rho}, \boldsymbol{C}, \boldsymbol{u}, p, N_{ele}$
Output: $\boldsymbol{res}$

1. ***int*** i = (thradIdx.x + blockIdx.x * blockDim.x);
2. ***if*** $(i < N_{ele})$
3. ***float*** $\rho_i = \rho(i)^p$;
4. ***int*** $id_1 \leftarrow 0$;
5. ***int*** $id_2 \leftarrow 0$;
6. ***for*** $\leftarrow j$ 0 ***to*** 8 ***do***
7. $id_1 \leftarrow read\ from\ \boldsymbol{C}$;
8. ***float*** $value \leftarrow 0$;
   // store result of multiplication
9. ***for*** $\leftarrow k$ 0 ***to*** 8 ***do***
10. $id_2 \leftarrow read\ from\ \boldsymbol{C}$;
11. $value \mathrel{+}= \rho_i * (K_e[j][k] * u[id_2])$;
    // result of multiplication
12. ***end***
13. atomicAdd (&$res[id_1]$, $value$);
    // atomic operation on GPU
14. ***end***

The global $thread\_id$ (i) in line 1 of Algo. 2 is the $global\_id$ of the finite element. The $thread$ reads the value of the elemental density of this element from the vector $\boldsymbol{\rho}$ and penalizes it by following the SIMP method, as shown in line 3. Next, in line 6, there is a loop over the nodes of this element. Inside the loop, the $thread$ loads the index id_1 from the connectivity matrix in line 7. It indicates the global position in the output vector, where the result of SpMV is to be written. The $thread$ initializes the variable $value$, which holds the result of the product in line 8. In line 9, there is another loop for all the nodes in the element. Here, another index id_2 is read from $[\boldsymbol{C}]$, which indicates the global position of vector $(\boldsymbol{u})$ with which the elements of $[K_e]$ are to be multiplied. Finally, the SpMV is computed and the result is multiplied with the penalized elemental density, as shown in line 11. When the $thread$ exits the innermost loop, the result of Mat-vec product for node j is stored in the variable $value$. The $thread$ ends by writing this result into the output vector $\boldsymbol{res}$, as shown in line 13.

It is to be noted that every node in the mesh is shared by multiple elements. It is possible that at a particular time, two or more threads are attempting to write the result at the same location, ultimately producing incorrect results. This problem is called a race-condition. In order to avoid race-condition during computation, atomic operation in CUDA is used. The final write operation to $\boldsymbol{res}$ in line 13 of Algo. 2 is an atomic add operation.

3 Results and Discussion

The proposed GPU-based topology optimization is tested on a 3D L-beam example. The design domain of the L-beam, along with its dimensions, supports, and load conditions, are shown in Fig. 2.

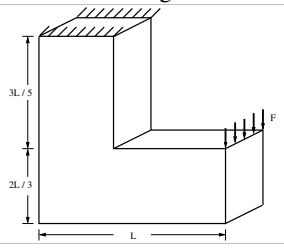


Fig. 2 Design domain of 3D L-beam

Table 1. The number of elements and nodes for the mesh sizes used.

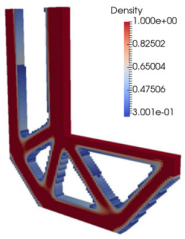
Mesh	No. of elements	No. of nodes
Mesh 1	32000	39606
Mesh 2	72000	88206
Mesh 3	103,680	126,582

The performance is analyzed over three different mesh sizes, as given in Table 1. For topology optimization, the final volume of the structure is constrained at $V^* = 50\%$ of the original volume. The other parameters are chosen as follows: Young's modulus $E = 1$, Poisson's ratio $\mu = 0.3$, penalty exponent $p = 3.0$, filter radius $= 4.0$, and the minimum density is 0.001 [5]. The termination criterion for CG is set as 10^{-3}, and the maximum no. of optimization iterations is 50.

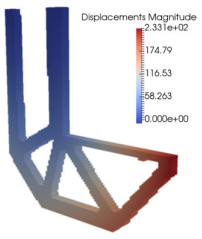
The CPU implementation is tested on a DELL Precision T3610 with Intel Xeon(R) CPU E5 1620; 3:70 GHz x 8, and 16 GB RAM. All the GPU experiments are performed on NVIDIA Tesla K40 card. It is equipped with 2880 CUDA cores and 12 GB of memory.

The final topology obtained after 50 iterations is shown in Fig. 3. As it can be observed from the figure that the main structure is composed of high-density elements, while the low-density elements are on the outer surfaces. The evolved topology is similar to the results shown in the study [22].

The total computation time per optimization iteration for both CPU and GPU over three mesh sizes are reported in Fig. 4. From the figure, it can be observed that for all the mesh sizes, the GPU implementation outperforms the CPU implementation. The ratio of the CPU and GPU time is referred to as the speedup, which is considered an important performance metric for parallel programming. The speedup with respect to the no. of elements is shown in Fig. 5. The speedup is seen to be varying between $2\times$ to $4\times$ for different mesh sizes.



(a) Densities in final topology



(b) Displacement field

Fig. 3. Final topology of the L-beam obtained after 50 optimization iterations.

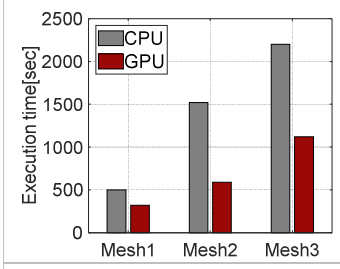


Fig. 4. Computation time for mesh sizes

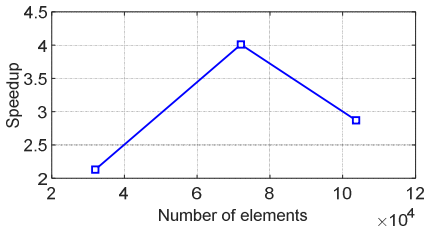


Fig. 5. Speedup for different mesh sizes

4 Conclusions

An implementation of SIMP-based topology optimization for a 3D linear elastic continua on GPU has been presented. In the implementation, the entire FEA computation has been performed on the GPU using the matrix-free CG solver. SpMV was performed using the EbE strategy, where each *thread* was assigned to one element of the finite element mesh. In order to avoid the race condition, the atomic operation was used. *Thrust* library of the CUDA toolkit was used for accelerating the linear algebraic operations. For performance analysis, an L-shaped beam was taken as an example with three different mesh sizes. Comparison to the CPU implementation showed $2\times$ - $4\times$ speedup for different mesh sizes. In future work, a more fine-grained parallel computing strategies [23, 24] can be used to increase speedup further.

References

1. Sharma, D., Deb, K.: Generation of compliant mechanisms using hybrid genetic algorithm. Journal of The Institution of Engineers (India): Series C, 95(4), 295-307(2014).
2. Aage, N., Lazarov, B. S.: Parallel framework for topology optimization using the method of moving asymptotes. Structural and multidisciplinary optimization, 47(4), 493-505(2013).
3. Kennedy, G., J., Martines, J. R.: Hybrid-parallel methods for large-scale gradient-based structural design optimization. 10th World Congress on Structural and Multidisciplinary Optimization (2013).
4. Bendsøe, M. P., Kikuchi, N.: Generating optimal topologies in structural design using a homogenization method. Computer methods in applied mechanics and engineering 71(2), 197-224(1988).
5. Schmidt, S., Schulz, V.: A 2589-line topology optimization code written for the graphics card. Computing and Visualization in Science, 14(6), 249-256(2011).
6. Wadbro, E., Berggren, M.: Megapixel topology optimization on a graphics processing unit. SIAM review, 51(4), 707-721(2009).
7. Duarte, L. S., Celes, W., Pereira, A., Menezes, I. F., Paulino, G. H.: PolyTop++: an efficient alternative for serial and parallel topology optimization on CPUs & GPUs. Structural and Multidisciplinary Optimization, 52(5), 845-859(2015).

8. Wu, J., Dick, C., Westermann, R.: A system for high-resolution topology optimization. IEEE transactions on visualization and computer graphics, 22(3), 1195-1208(2015).
9. Suresh, K.: Efficient generation of large-scale pareto-optimal topologies. Structural and Multidisciplinary Optimization, 47(1), 49-61(2013).
10. Zegard, T., Paulino, G. H.: Toward GPU accelerated topology optimization on unstructured meshes. Structural and multidisciplinary optimization, 48(3), 473-485(2013).
11. Martínez-Frutos, J., Herrero-Pérez, D.: Large-scale robust topology optimization using multi-GPU systems. Computer Methods in Applied Mechanics and Engineering, 311, 393-414(2016).
12. Martínez-Frutos, J., Martínez-Castejón, P. J., Herrero-Pérez, D.: Efficient topology optimization using GPU computing with multilevel granularity. Advances in Engineering Software, 106, 47-62(2017).
13. Challis, V. J., Roberts, A. P., Grotowski, J. F.: High resolution topology optimization using graphics processing units (GPUs). Structural and Multidisciplinary Optimization, 49(2), 315-325(2014).
14. Martínez-Frutos, J., Herrero-Pérez, D.: GPU acceleration for evolutionary topology optimization of continuum structures using iso-surfaces. Computers & Structures, 182, 119-136(2017).
15. Ram, L., Sharma, D.: Evolutionary and GPU computing for topology optimization of structures. Swarm and Evolutionary Computation, 35, 1-13(2017).
16. Kiran, U., Sharma, D., Gautam, S. S.,: GPU-warp based finite element matrices generation and assembly using coloring method. Journal of Computational Design and Engineering, 6(4), 705-718(2019).
17. Sanfui, S., Sharma, D.: Exploiting Symmetry in Elemental Computation and Assembly Stage of GPU-Accelerated FEA. In: Liu, G. R., Cui, F., Xiangguo, G. X. (eds.) 10th International Conference on Computational Methods (ICCM2019), 9–13 July 2019, pp. 641 – 651, ScienTech Publisher ,Singapore.
18. Kiran, U., Sanfui, S., Ratnakar, S. K., Gautam, S. S., Sharma, D.: Comparative Analysis of GPU-Based Solver Libraries for a Sparse Linear System of Equations. In Advances in Computational Methods in Manufacturing, pp. 889-897, Springer, Singapore(2019).
19. Nvidia, C.U.D.A.: Nvidia CUDA C programming guide. Nvidia Corporation, 120(18), 8(2011).
20. Bell, N., Hoberock, J.: Thrust: A productivity-oriented library for CUDA. In GPU computing gems Jade edition, 359-371(2012). Morgan Kaufmann.
21. Bendsoe, M. P., Sigmund, O.: Topology optimization: theory, methods, and applications. Springer Science & Business Media (2013).
22. Valdez, S. I., Botello, S., Ochoa, M. A., Marroquín, J. L., Cardoso, V.: Topology optimization benchmarks in 2d: Results for minimum compliance and minimum volume in planar stress problems. Archives of Computational Methods in Engineering, 24(4), 803-839(2017).
23. Subhajit Sanfui and Deepak Sharma, "GPU Acceleration of Local Matrix Generation in FEA by Utilizing Sparsity Pattern", In 1st International Conference on Mechanical Engineering (INCON 2018), 4–6 January 2018, Jadavpur University, India.
24. Subhajit Sanfui and Deepak Sharma, "A Two-Kernel based Strategy for Performing Assembly in FEA on the Graphic Processing Unit", In IEEE International Conference on Advances in Mechanical, Industrial, Automation and Management Systems, 3-5 February 2017, MNNIT Allahabad, India.