Stefan Gavranovic

GPU Accelerated Multi-Physics FEM Solver for Topology Optimization

I. Introduction

Additive manufacturing is driving a revolution in manufacturing. With the help of such technology, objects can be produced by successively adding thin layers of material. Nowadays, this procedure is used to create a wide variety of goods such as: customized medical devices, dental and hip implants or even hearing aid devices. Significant application areas of additive manufacturing are found in aerospace industry. Outcome is that less material is being used compared to conventional manufacturing techniques. Therefore, the production costs are being reduced, significant amounts of fuel are saved, which in turn implies reduction in greenhouse gases emission.

Since additive manufacturing is offering to designers exploration of nearly infinite design spaces, the importance of topology optimization is evergrowing. Topology optimization is a mathematical method for optimal placement of a material within a given design space, boundary conditions, and loads, in order to satisfy the desired objectives.

In a stage of rapid prototyping, topology optimization can assist engineers to create solutions that meet design requirements, having optimal material usage while not compromising structural integrity. In recent years a lot of research was invested in exploring and establishing the theory of topology optimization. The application field of topology optimization has expanded beyond structural mechanics, now including fluid flows, acoustics, heat transfer, and many more. However, most of the research has been carried out for 2D models. Due to high computational costs, performing topology optimization for 3D models may require hours, or in some cases even days, which hinders rapid prototyping design process. Ideally, a designer would like to have almost instantaneous feedback when exploring a design space. Not much research has been carried out with a goal to improve computational efficiency, as it was done for establishing the theory of topology optimization. Therefore, in this work, use of multi-core architecture such as GPU (Graphics Processing Unit) combined with efficient numerical algorithms is demonstrated.

Resume:

With development of additive manufacturing, the importance of topology optimization is evergrowing. With the help of topology optimization engineers and designers can create better products while using less material. By doing so, waste and energy consumption is being reduced, contributing to overall reduction of greenhouse gas emissions. In this work a fast finite element solver with application to topology optimization is presented. Combining efficient numerical algorithms and data structures such as geometric multigrid method and hexahedral meshes, with the state of the art many-core architectures results in nearly interactive topology optimization process.

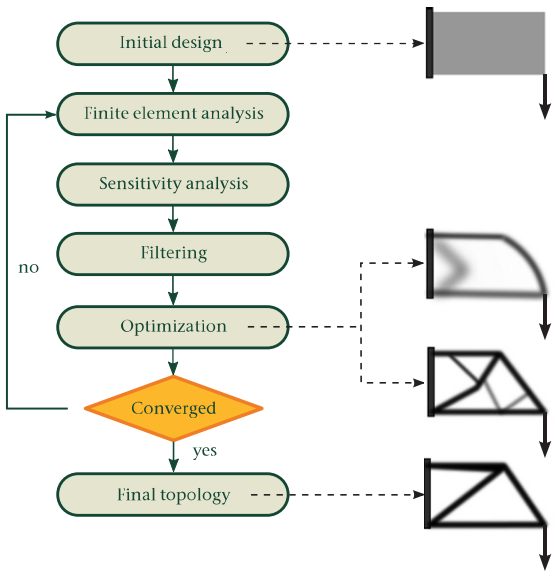


Figure 1:
Topology Optimization algorithm.

II. Topology Optimization

Algorithmic representation of the topology optimization process with its key components is graphically presented in the *Figure 1*. Starting from the initial design, given that loading conditions are known, displacement and stress analysis is performed. This valuable information is then used to calculate sensitivity of the structure to a change in material layout. In order to ensure existence of a solution, sensitivities must be filtered. Finally, in the optimizer material layout is redistributed according to the previously calculated sensitivity information. This process is repeated in an iterative fashion, until there is no change in the material layout.

Previously mentioned material layout is controlled with the help of pseudo densities. These pseudo densities are design variables assigned to each eleemnt of the computational domain and with permissible values $\rho \in [0, 1]$. Element carring value of pseudo density $\rho = 1$ signifies material, and $\rho = 0$ signifies void.

Mathematically formal way of expressing this optimization problem is given by the following set of Equations:

$$\begin{aligned} \underset{\rho}{\text{minimize}} \quad & c(\rho) = \mathbf{u}^\top \mathbf{K}(\rho)\mathbf{u} \\ \text{subject to} \quad & \frac{V(\rho)}{V_0} = \alpha \\ & \mathbf{K}(\rho)\mathbf{u} = \mathbf{f} \\ & 0 < \rho_{min} \leq \rho \leq 1. \end{aligned}$$

where objective function $c\ (\rho)$ function is minimized and it is subjected to a volume constraint of a given volume fraction α, being the ratio between the material volume $V\ (\rho)$ and the design domain volume V_0. Displacement and force vectors are denoted respectively $\mathbf{u}$ and $\mathbf{f}$. Matrix denoted as $\mathbf{K}(\rho)$ describes the underlying physical system. For solving the aforementioned problem, gradient based optimization methods are usually being used.

III. Fast Solver

Since the optimization process comes at a high computational cost of performing the finite-element method (FEM) analysis at each optimization step, the main focus of this work is to develop an efficient solver for performing FEM analysis.

At this stage of the topology optimization, displacements and stresses of the structure subjected to loads are being calculated. Traditionally, a new computational mesh would be created at every iteration of the optimization process, and whole linear system of equations would be newly assembled and solved. In the case of highly detailed simulation, number of underlying linear equations can be in order of millions. This of course, reflects negatively on the computational time necessary to perform topology optimization.

In order to be able to solve problems on a large scale fast, on every iteration step, following key building blocks are considered:

- **Structured Mesh** – computations are performed on structured meshes where every element is of the same dimension.
- **Geometric Multigrid (GMG)** Solver – very efficient linear algebra solver that is dependent on the computational mesh.
- **Immersed Boundary Treatment** – boundaries of a computational domain are embedded within mesh and accounted through model equations.

Structured Mesh

In order to perform finite element analysis, computational domain must be discretized into elements. During this meshing process, standard procedure is to fit the shape of each element to the boundary of a computational geometry as it is represented in the *Figure 2b*.

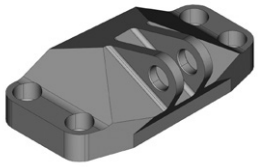
Fitting tetrahedral elements to a geometry might require a lot of computational effort. This, of course depends on the shape complexity of the geometry that is being analyzed. In practice, computational geometries might have very complex shapes, resulting with meshing that can take hours to complete.

Alternatively, geometry can be discretized with large number of uniform hexahedral elements, each having the exact same dimenstion. Some of the main advantages of using hexahedral elements are:

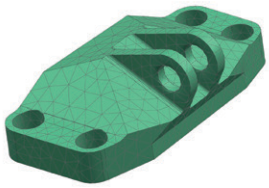
- Uniformity of computation – since all elements are of identical dimensions, same computational instructions can be applied to each element.
- No need for re-meshing – regardless of complexity of a geometry, mesh
- Data structure simplicity – a lot of element specific data can be precalculated.

Due to the uniformity of the mesh, obtaining voxels that are embedding the geometry is done in a parallel way using GPU accelera-

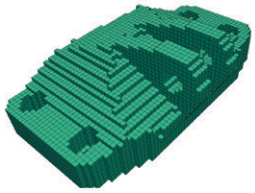
Figure 2:
Different methods of obtaining computational mesh.



(a) Computational geometry.



(b) Tetrahedral mesh of the computational geometry.



(c) Hexahedral mesh of a computational geometry.

tion. Massive parallelism capability of GPUs allow creating large meshes consisting of millions of elements in a matter of milliseconds.

Immersed Boundary Treatment

Major drawback of structured hexahedral-type mesh is its inability to accurately represent geometry which can be seen in the *Figures 2c, and 3*. It can be noticed that geometry is not preserved, rather it is embedded into the hexahedral mesh. This inacuracy in discretization leads to degradation of result accuracy. It is very important to be able to control such error. In order to do so, immersed boundary methods are used.

Figure 6 has been removed and changed the numbers of the figures from this point

Immersed boundary methods are used in order to enforce correct values at the nodes of a hexahedral mesh, such that their interpolation to the true geometry provides accurate results. This effect is achieved by augmenting the system matrix $\mathbf{K}(\rho)$ with additional term $\mathbf{P}(\rho)$ stemming from the variational formulation of the problem, as shown in the Equation 1.

$$(\mathbf{K}(\rho) + \mathbf{P}(\rho))\mathbf{u} = \mathbf{f}$$

This augmented system is then being solved in order to provide more accurate results.

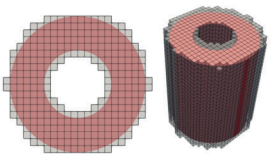


Figure 3:
Cylinder geometry embedded withing the hexahedral mesh.

Geometric Multigrid Solver

One of the most efficient numerical methods for solving systems of linear equations are multigrid (MG) methods. It can be theoretically shown that multigrid methods have optimal complexity O(N) for solving systems of linear equations of N unknowns for some elliptic types of differential equations. Two major types of multigrid methods are to be differentiated: arithmetic multigrid (AMG) and geometric multigrid (GM).

Arithmetic multigrid methods are not dependent on the computational domain discretization. This property makes arithmetic multigrid methods suitable as black-box solvers. On the other hand, geometric multigrid methods depend on the geometry of the mesh and often are not suitable for unstructured meshes. When working with structured hexahedral meshes, geometrical multigrid comes as a natural choice. Therefore, it represents the core method of this work.

Multigrid methods are based on creating hierarchies of grids starting from the grid on which differential equations are discretized. This grid can be noted as Ω_h, where h represents a characteristic length of a grid element, in this case the side of a hexahedral element. Each consecutive coarser grid is constructed by doubling the characteristic length of the previous grid. Hence, the

preceding grid to the Ω_h can be denoted as Ω_{2h}. In the *Figure 4a* the process of a grid construction for a 2D case is demonstrated. An element of the grid Ω_h is considered to be active if it contains at least some part of the computational geometry. If there exists at least one active element on Ω_h which is covered by the element on the Ω_{2h}, then that coarse grid element is active as well. This coarsening strategy continues until the coarsest grid $\Omega_{2^n h}$ is reached, where n denotes the number of levels in the multigrid hierarchy.

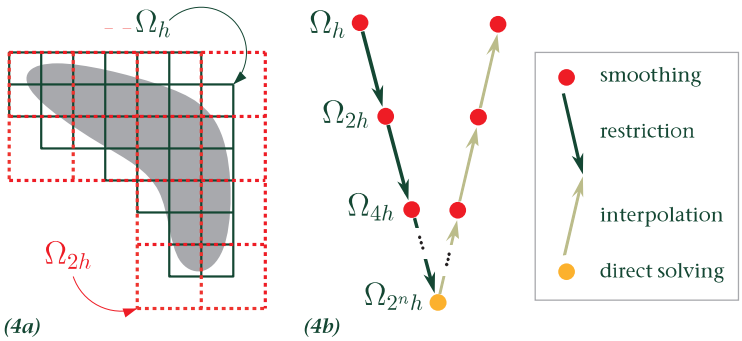


Figure 4:
Grid hierarchies:
(a) *Green represent active elements of the grid h, while red squares represent active elements of the coarser grid Ω_{2h}.*
(b) *Mesh coarsening in V-cycle*

The main goal of the multigrid method is to obtain solution on the finest mesh level, by applying corrections transferred from the solutions on the coarser mesh levels. This procedure is repeated in an iterative fashion as shown in the *Figure 4b*.

GPU Parallelization

Graphics Processing Unit (GPU) is a single-chip processor mostly used to accelerate the rendering of video and graphics content. With the significant increase in computational performance, over the past few years, GPUs became important source of extra compute power in the field of scientific computing.

Some of the main differences between CPU and GPU architecture are:

- a GPU contains thousand of cores that can handle thousand of threads simultaneously while CPU has only a few cores with a lot of cache memory that can deal with only a few threads at the same time *(see Figure 5)*;
- CPU cores are optimized for sequential serial processing, while GPU cores are designed to handle multiple tasks in parallel.

Carefully selected set of previously mentioned algorithms and data structures, allows easy parallelization using GPUs, therefore achieving significant speedup compared to CPU execution times.

IV Results

One of the benchmarks for above mentioned combination of algorithms is tractor

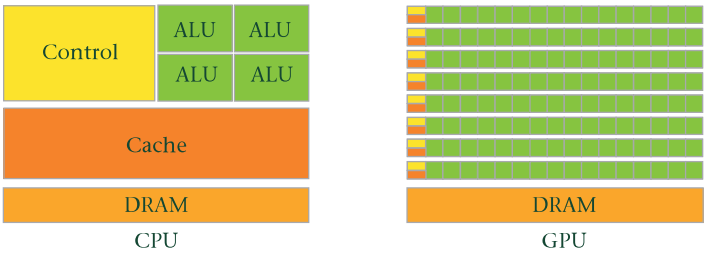


Figure 5 (left):
CPU vs. GPU. GPU uses more resources for data processing than for flow control and cache operations.
This Figure was created on the basis of illustration provided by NVIDIA.

lever part. Position of the part in the assembly and its loading conditions have been displayed in *Figure 6*. Optimization has been performed with respect to both loading cases presented in the *Figure 6a*.

Another standard benchmark geometry that is often being used to evaluate performance of topology optimization tools is the jet engine bracket shown in the *Figure 7*. All tests were performed using machine equipped with an Intel Core i7-6900K CPU and NVidia TITAN X(Pascal) Graphics card.

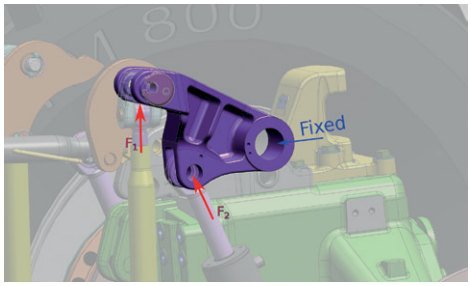
In the *Table 1* results of the optimization cases are presented. First column represents the geometry that is being optimized. Next column denotes the number of elements used to discretize optimization domain. Following two columns present number of iterations until convergence to the solution with given weight reduction percentage. Finally, last two columns represent time necessary to perform single FEM analysis for all the load cases of a given model per iteration, and a total time necessary for topology optimization process to finish.

V Conclusion and Future Work

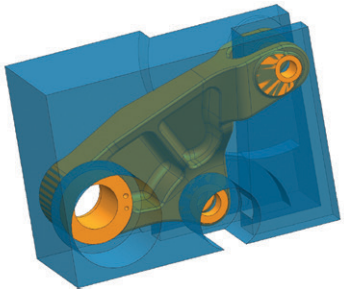
Fast simulation coupled with topology optimization provides an opportunity for designers to obtain prototypes very fast and therefore create better and more energy efficient products.

Table 1: *Optimization parameters and runtime.*

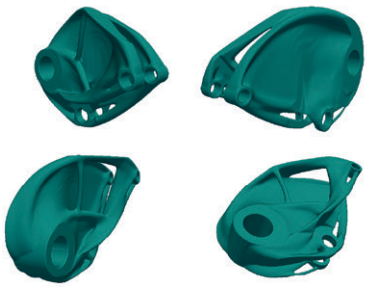
Geometry	Elements	Iterations	Volume Reduction	FEM/ iter (s)	Total Optimization Runtime (s)
Lever	1.3×10^6	*37*	*80 %*	*1.6*	*135.3*
Lever	0.47×10^6	*32*	*80 %*	*0.6*	*54.32*
Bracket	1.5×10^6	*35*	*75 %*	*2.1*	*154.32*
Bracket	0.47×10^6	*31*	*75 %*	*0.9*	*74.32*



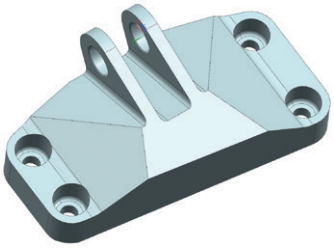
(6a)



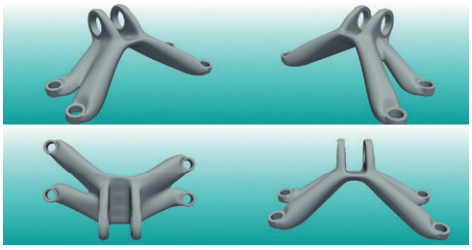
(6b)



(6c)



(7a)



(7b)

Figure 6:
Optimization of the tractor lever part:
(a) *Part is subjected to 2 load cases*
(b) *Part within permissible optimization domain*
(c) Optimized part with 80 % weight reduction compared to the design domain

Figure 7:
Initial and optimized jet engine bracket geometries.
(a) *Jet engine bracket.*
(b) *Optimized bracket geometry with 75% weight reduction*

Presented examples show that the use of efficient algorithms and data structures such as structured mesh, geometric multi-grid solver, and GPU acceleration can deliver optimized shapes within several seconds. In comparison, most of the commercially available tools on the market require hours to compute same benchmarks with comparable quality of the results. Future work will be focused on further improvement of computational efficiency by the means of adaptive mesh refinement. Additionally, use of fast solver within topology optimization for different physics will be investigated.