Efficient hybrid topology optimization using GPU and homogenization-based multigrid approach

Arya Prakash Padhi1 · Souvik Chakraborty2,3 · Anupam Chakrabarti4 · Rajib Chowdhury4

Received: 28 May 2022 / Accepted: 1 December 2022 / Published online: 29 December 2022
© The Author(s), under exclusive licence to Springer-Verlag London Ltd., part of Springer Nature 2022

Abstract
We propose an efficient implementation of a new hybrid topology optimization algorithm based on multigrid approach that combines the parallelization strategy of CPU using OpenMP and heavily multithreading capabilities of modern Graphics Processing Units (GPU). In addition to that, significant computational efficiency in memory requirement has been achieved using homogenization strategy. The algorithm has been integrated with versatile computing platform of MATLAB for ease of use and customization. The bottlenecking repetitive solution of the state equation has been solved using an optimized geometric multigrid approach along with CUDA parallelization enabling an order of magnitude faster in computational time than current state of the art implementations. The main novelty lies in the efficient implementation wherein on the fly computation of auxiliary matrices in the multigrid scheme and modification in interpolation schemes using homogenization strategy removes memory limitation of GPUs. Memory hierarchy of GPU has also been exploited for further optimized implementations. All these enable solution of structures involving hundred millions of three dimensional brick elements to be accomplished in a standard desktop computer or a workstation. Performance of the proposed algorithm is illustrated using several examples including design dependent loads. Results obtained indicate the excellent performance and scalability of the proposed approach.

Keywords Topology optimization · MultiGrid · Homogenization · CUDA GPU acceleration

1 Introduction
Topology optimization attempts to find the efficient distribution of materials within a design domain under specific loading and boundary conditions. Unlike size and shape optimization, topology optimization allows for the creation of a material distribution without the need for a predetermined structural arrangement. Since Bendsoe and Kikuchi’s [1] early work on structural topology optimization, it has gained significant momentum in the past decades and has been successfully applied to automotive, aerospace, thermal, structural, energy and meta material applications [2–7]. Several semi-analytical to numerical techniques (such as density based methods, level set, topological derivative, moving morphable methods) have been developed to incorporate simple compliance minimization to the more complex multi-physics and multi-material topology optimization [8] using finite element or isogeometric analysis solution strategy [9–11]. In spite of many advances, computing continues to be a key impediment especially for large structures requiring finer details which usually takes several days (for example in [12]) even in HPC systems. As an alternative to conventional CPU based parallel computing, graphics processing units (GPU) based computing has recently emerged as a viable alternative. The goal of this paper is to exploit recently developed GPU hardware, classical CPU based parallel computing (OpenMP) and multigrid (MG) methods to develop an efficient topology optimization tool.

Some work on development of GPU based topology optimization algorithm has already been carried out in the literature.
Wadbro and Berggren [13] advocated using GPU computation to evaluate high-resolution finite element models in heat conduction topology optimization. To decrease device memory needs, they used a Preconditioned Conjugate Gradient (PCG) technique with an assembly-free element-wise implementation. Researchers have also developed GPU accelerated solvers for multigrid preconditioned conjugate gradient solvers over past few years [14, 15]. Schmidt and Schulz [16] suggested a GPU-based node-wise assembly-free PCG solution for addressing elasticity issues at every iteration of the structural compliance minimization problem. The iterative solver’s matrix-vector operations provided significant speed up by utilizing shared memory in the proposal. Reducing the grain size in the assembly-free GPU implementation is another way to boost the computational speed [17]. GPU acceleration has also been used in [18] for level set method (LSM) and isogeometric analysis (IGA) based topology optimization.

One of the primary bottlenecks associated with GPU is its limited memory. In this context, MG techniques provide a viable alternative owning to its multiscale nature. In structural mechanics Krylov subspace techniques frequently utilize these approaches as a preconditioner. Dick et al [19] developed an efficient node-wise matrix-free GPU implementation of the geometric multigrid technique with stencil computing for elasticity problems. To avoid storing the coefficient matrix, they employ a Cartesian grid and parallel GPU computation. By utilizing a stencil algorithm, data locality may be used as well. This enables the merging of memory access into a single memory transaction and provides fast stencil-based “on the fly” grid transfer operators. This idea has been reflected in [20, 21]. Other notable work in this area includes [16, 17, 19, 22–24].

In spite of the research articles discussed above, efficient GPU implementation for topology optimization has still remained a challenging field due to asynchronous computational nature of CPU and GPU, memory size and bandwidth limitation of GPUs, and non availability of software tools for easy integration and customization; the objective of this paper is to address some of these limitations. In this paper, we propose a novel hybrid scheme that exploits the optimal capacity of both GPU and CPU by appropriate distribution of the computations. Memory constraint associated with GPU computation is addressed by using a novel geometric multigrid preconditioner. To further accelerate the algorithm, a homogenization scheme has also been used. The algorithm proposed is converted into a software package and integrated with recently developed base framework [25] as a custom module. We illustrate the efficacy of the developed framework on number of high dimensional topology optimization problems involving millions of design variables. Additionally, the developed module can be seamlessly integrated with any existing topology optimization routine.

The rest of the paper is organized as follows. In Sect. 2 the standard density based topology optimization algorithm is described. In Sect. 3 multigrid pre-conditioned conjugate gradient approach is explained. In the next section its efficient hybrid implementation is detailed. Subsequently several numerical experiments are carried out in Sect. 5. Finally, important observations are summarized in Sect. 6.

## 2 Density-based topology optimization

A binary programming problem, topology optimization aims to find the optimal material layout (solid and void) that minimizes an objective function [26]. When designing a material layout, it’s important to adhere to a set of design constraints. As a result of their conceptual simplicity, density-based methods are the most widely used topology optimization methods in commercial or industrial software. Usually the design domain is discretized for two-fold benefit. First one is to update the density of each discretized element independent of others and the second one is to use the above discretization scheme to compute response of the system using finite element methods. It is possible to formulate the topology optimization problem as follows [27]:

\[
\min_{\rho} \quad c(\rho, u) \\
\text{s.t.} \quad K(\rho)u = f \\
\quad V(\rho) \leq V_0 \\
\quad 0 \leq \rho(x) \leq 1, x \in \mathbb{R},
\]

where \(c\) is the cost function, \(\rho\) denotes density design variables, \(u\) represents the response of the system, \(K\) is the global stiffness matrix, \(f\) is the force vector, and \(x\) is array of discretized elements. The design domain is demarcated by \(\mathbb{R}\) and the target volume of optimized shape \(V(\rho)\) must be smaller than a prescribed value \(V_0\). The unknown density parameters, \(\rho(x)\), are utilized to adjust the finite element’s stiffness in the regular mesh. Although discrete densities are desirable, use of continuous form helps in easy gradient computation and smooth transition at boundaries. In reality, this parameterization results in design with huge regions of intermediate densities that, while numerically ideal, are impractical to produce [28]. Hence the density is modified to an artificial density form for computational convenience and drive the solution to binary 0 (void) or 1 (solid material). Typically, this problem is handled utilizing implicit relaxation/penalization approaches, which force the topology design towards solid/void topology. The solid isotropic material with penalization (SIMP) approach employs implicit penalization techniques through a power-law interpolation function between void and solid to calculate the stiffness matrix of each element \(K_e\) similar to [29] as follows:

\[
K_e = K_{\text{min}} + \rho_e^p(K_0 - K_{\text{min}}),
\]

where \(K_0\) corresponds stiffness matrix when an element is fully solid and \(K_{\text{min}}\) corresponds to minimum stiffness for
least allowable density of an element. The later is provided to avoid singularity issue. Even though the use of material interpolation scheme allows for the creation of designs which are almost solid and void, they destroy the optimization problem’s convexity thereby, increasing the risk of ending up in a local minimum. As a result, continuation methods are commonly used when solving optimization problems in order to avoid premature convergence to local minima. Continuation-based methods take “global” information into account and are more likely to ensure “global” convergence, or at the very least, convergence to better designs [30].

To prevent numerical challenges and modelling issues such as mesh-dependency of solution and checker-board patterns, the topology optimization problem should also be regularized utilizing extra density field constraints. The sensitivity filter is employed in this study since it has been demonstrated in practice to be successful in providing mesh-independent solutions [31]. Furthermore, gradient filtering is motivated by continuum mechanics and may favour convergence of particular length scales over others, therefore hastening convergence. The sensitivity filter provides computational advantages because it is not included in the optimality criteria (OC) updating scheme loop.

The sensitivity filter, as shown below, adds some kind of smoothing on the derivatives of the objective function as follows:

\[
\frac{\partial c}{\partial \rho} = \frac{\sum_{i \in \text{NB}_e} w(x_i, x_e) \rho_i \frac{\partial c}{\partial \rho}}{\max (\gamma, \rho_e) \sum_{i \in \text{NB}_e} w(x_i, x_e)}
\] (3)

where \(\text{NB}_e\) is an element’s neighbourhood set, \(w(x_i, x_e)\) is a weighting function, and \(\gamma > 0\) is a small number to prevent division by zero. In present work, the weighting function is defined as:

\[
w(x_i, x_e) = \begin{cases} r - ||x_i - x_e|| & \text{if } ||x_i - x_e|| \leq r \\ 0 & \text{if } ||x_i - x_e|| \geq r \end{cases}
\] (4)

while an element’s neighbourhood is defined as:

\[\text{NB}_e := \{i \mid \text{dist}(i, e) \leq r\},\] (5)

where \(r\) is the filter radius and \(\text{dist}(i, e)\) emphasizes that it includes all elements \(i\) within the distance \(R\) from the center of element \(e\).

Although SIMP can be used for solving a wide array of problems including heat sinks and other multi-physics problems [32, 33], we are interested in minimization of structural compliance,

\[c = f^T u,\] (6)

where \(f\) is the applied force vector and \(u\) is the corresponding displacement vector.

Considering the discretized linear state system \(Ku = f\) the sensitivities of Eq. (6) using adjoint state method (solving for \(u^*\) in \(Ku^* = \frac{\partial f}{\partial u}\)) with respect to \(\rho\), we obtain

\[
\epsilon_{\rho} = \frac{\partial c}{\partial \rho} = -u^T \frac{\partial K}{\partial \rho} u
\] (7)

\[= -u^T (p \rho^{\eta-1} (K_0 - K_{min})) u.
\]

where \(p\) is a penalty factor (usually 3 but can be obtained more accurately by continuation methods) and \((\cdot)^T\) is the transpose of the vector. These sensitivities given by Eq. (7) permit to update the design variables \(\rho\) using some sequential convex approximations, such as Sequential Quadratic Programming (SQP) [34] or Method of Moving Asymptotes (MMA) [35]. The Optimality Criterion (OC) updating scheme proposed by [36] is adopted in this work due to its numerical efficiency. The OC updating scheme is as follows:

\[
\rho_{e+1} = \begin{cases} \max\{(1 - m), 0\} & \text{if } \rho_{e} B_{e}^* \leq \max\{(1 - m), 0\}, \\
\min\{(1 + m), 1\} & \text{if } \min\{(1 + m), 1\} \leq \rho_{e} B_{e}^* \\
\rho_{e} B_{e}^* & \text{otherwise},
\end{cases}
\] (8)

where \(m\) is a positive move-limit, \(\eta\) is a numerical damping coefficient (usually \(\eta = 1/2\), \(q\) is a penalty factor to further achieve black-and-white typologies (typically \(q = 2\)) and \(B_{e}^*\) is the Karush–Kuhn–Tucker (KKT) optimality condition. The Lagrange multiplier \(\lambda\) is found using bisection method. The algorithm stops when maximum number of iterations is reached or when the change in the variable \(\rho_{e+1} - \rho_{e}\) fall below a prescribed value.

The topology optimization pipeline’s main bottleneck is the solution of the first constraint in Eq. (1) which is typically calculated using finite element analysis (FEA). This stage entails two computationally demanding tasks: assembling the local element equations into a global system of equations and solving that resultant linear system. These computationally expensive tasks may result in an unsustainable situation in terms of computing time and memory usage. This issue is more prominent when working with large-scale models [37] or when the system response must be re-evaluated again and again, as in topology optimization. Iterative solvers and assembly-free techniques have been widely utilized to reduce FEA memory requirement at the expense of increasing the processing time of the solution step, which is usually eased by parallel computing. This is explained in the next section.
3 Multigrid pre-conditioned approach

3.1 Pre-conditioned Conjugate Gradient (PCG) Method

The equation to be solved in the optimization loop discussed earlier is of the form of:

\[ Ku = f \] (10)

Efficient direct solution method involves cholesky or LU decomposition, where matrix \( K \) is decomposed into a lower triangular and an upper triangular matrix system and subsequently solved using forward and back substitution staying away from matrix inversion all throughout. However, for large systems, this is still prohibitive and inefficient especially when matrix \( K \) is very large and a sparse one [38]. The conjugate gradient technique is a mathematical methodology for numerically solving system of linear equations, specifically those with positive-definite matrices. It essentially solves Eq. (10) but as a minimization problem of the following alternate quadratic form:

\[ f(x) = \frac{1}{2}u^T Ku - u^T f. \] (11)

For positive definite system matrices (which is actually the case in many natural physical phenomena), the traditional Conjugate Gradient technique is the preferred iterative method [39]. It is used to minimize the functional \( F(u) = \| Ku - f \|_{K^{-1}} \) by multiplying the matrix vector just once in each iteration. As a matter of fact, this approach can theoretically arrive at the answer in less than \( n \) iterations. In-fact the convergence rate can be given by,

\[ \| u - u_k \|_{K^{-1}} \leq \| u - u_0 \|_{K^{-1}} \left( \frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1} \right)^k \] (12)

where \( \kappa \) is the matrix \( K \)'s condition number, and \( k \) is the number of iterations. It takes a long time for the system to reach convergence for \( \kappa >> 1 \). Hence the original equation is usually modified for improved convergence by pre-multiplying both sides of Eq. (10) by \( M^{-1} \).

\[ M^{-1} Ku = M^{-1} f, \] (13)

where \( M \) is a matrix or an operator such that \( \kappa(M^{-1}K) << \kappa(K) \), and it is used to precondition the linear system in order to assure and accelerate convergence. There is a minimal expense involved in building an effective preconditioner \( (M^{-1}) \) and the condition number should be as near to unity as feasible and independent of the number \( n \). Incomplete Cholesky factorization, diagonal scaling, and Factorized Sparse Approximate Inverses (FSAI) are examples of classical preconditioners that do not offer mesh-independent convergence rates. In this study, contemporary multilevel/multigrid approach which is numerically scalable is used instead (e.g., [40]). The PCG algorithm is depicted in algorithm 1. To ensure enough accuracy the tolerance value in the PCG loop is taken to be \( 10^{-8} \).

Algorithm 1: Preconditioned Conjugate Gradient Method

1. Initialization: \( x_0 = 0, r_0 = b, z_0 = M^{-1}r_0 = 0, k = 1, p_0 = z_0, \alpha_0 = r_0^T r_0 / (p_0^T Ap_0) \)

2. while \( r_k^T r_k > tol \) do

3. \( k = k + 1 \)

4. \( x_{k+1} = x_k + \alpha_k p_k \)

5. \( r_{k+1} = r_k - \alpha_k Ap_k \)

6. \( z_{k+1} = M^{-1} r_{k+1} \)

7. \( \beta_{k+1} = r_{k+1}^T z_{k+1} / (r_k z_k) \)

8. \( p_{k+1} = z_{k+1} + \beta_{k+1} p_k \)

9. \( \alpha_{k+1} = r_{k+1}^T z_{k+1} / (p_{k+1}^T Ap_{k+1}) \)

10. end

11. return \( x_{k+1} \)
Multigrid Preconditioned Conjugate Gradient (MGCG) method has been successfully used in the discretized finite element solution part to solve the state space equation. This high-efficiency iterative technique has been used for solving large-scale linear equation because of its highly efficient preconditioning technique, faster convergence and minimal computational effort involvement and scalability. The multigrid method upon which the preconditioner is built is described below.

### 3.2 Multigrid method (MG)

Solution of the linear system of equation $Ku = f$ by classical iterative scheme, is generally done by resolving $K$ into matrices $M$ and $N$ with non-singular $M$, such that $K = M - N$. Thus,

$$Mu = Nu + f$$

or

$$u = \left( M^{-1}N \right) u + M^{-1}f \tag{14}$$

Given an initial iterate $u^{(0)}$, a fixed point iteration can be applied to this equation

$$u^{(m+1)} = Su^{(m)} + M^{-1}f, \quad m = 0, 1, 2, 3, \ldots \tag{15}$$

This basic iterative approach might also be damped with damping coefficient $\omega$:

$$u^* = Su^{(m)} + M^{-1}f, \quad u^{(m+1)} = \omega u^* + (1 - \omega)u^{(m)} \tag{16}$$

such that

$$u^{(m+1)} = (\omega S + (1 - \omega)I)u^{(m)} + \omega M^{-1}f \tag{17}$$

If $u$ is the actual solution of the original equation and $u^{(m)}$ is the approximation computed using above, the error is denoted by,

$$e^{(m)} = u - u^{(m)} \tag{18}$$

and the residual is given by

$$r^{(m)} = f - Au^{(m)} = Ae^{(m)} = r^{(m)} \tag{19}$$

When $M = \text{diag}(A) = D$ in Eq. (14), a straightforward calculation can show that, the iterative solution is of the form:

$$u^{(m+1)} = u^{(m)} + D^{-1}r^{(m)} \tag{20}$$

This is known as the Jacobi method. With damping coefficient $\omega$, the damped Jacobi method can be written as:

$$u^{(m+1)} = u^{(m)} + \omega D^{-1}r^{(m)}, \quad \omega \in (0, 1]. \tag{21}$$

In multigrid (MG) methods, the residual calculation step (Eq. (19)) is used for updating the current iterate $u^{(m)}$. An approximation $\tilde{e}^{(m)}$ of $e^{(m)}$ is computed from Eq. (20) and the new iterate is given by $u^{(m+1)} = u^{(m)} + \tilde{e}^{(m)}$.

Another important component of MG method is the grid transfer mechanism. A simple way for the same is depicted below in Fig. 1 in which a uniform refinement is called restriction (denoted by $l^h_{2h}$) and that from $\Omega^h$ to $\Omega^{2h}$ is called prolongation or interpolation (denoted by $l^h_{2h}$).

The iterative single level methods (e.g Jacobi solution) can quickly reduce the high frequency components
of the mistake, but it performs poorly for low frequency errors. The MG technique is well known for being one of the most efficient strategies to enhance the convergence rate. The goal of MG is to build multiple grids at different scale (resolution) of discretization. Then, at each level, repeated relaxations are performed to reduce high-frequency errors on tiny grids and low-frequency errors on coarse grids. A linear system can be solved with the help of MG at a cost of $O(n)$. Smoothing and coarse-grid correction are two complimentary procedures that work together to produce optimal performance. Gauss-Seidel or Jacobi stationary iterative methods are commonly used to smoothing out the solution and decreasing oscillation errors. The combination of smoothing, restriction, and prolongation works very well and results in converged solution. A simple two grid algorithm is depicted in algorithm 2. This algorithm basically transfers the equation from fine grid of $\Omega^h$ domain to the two times coarser grid of $\Omega^{2h}$, solves it there and interpolates back to the finer grid (Figs. 2, 3).

The important question now, is the solution in coarser grid which is step 4 of algorithm 2. Looking carefully it is again essentially a linear system and can be solved in a manner we started solving in algorithm 2. This necessitates the repeated application of two-grid algorithm and gives rise to recursive hierarchical MultiGrid solution. This typical flow of processes is shown in Fig. 4, which is commonly known as V-cycle. The corresponding algorithm is written in algorithm 3. This approach splits the grids into multiple sizes and computes the precise answer only at the coarsest grid corresponding to the largest discretization size. In this multilayer mesh method, the finest mesh level is denoted by $l$ (level) = 1, whereas $l$ (level) = $L$ represents a coarser level as we can see in Fig. 5. This popular V-cycle method is utilized in this study. The combination of multigrid with conjugate gradient method simplifies the computational complexity of the problem significantly and its comparison with direct solver with number of degrees of freedom $N$ is shown in Table 1. Storage requirement as well as computational time is proportional to $N$ in mgcg but in direct solver it is $N\sqrt{N}$. In our study we have used total number of levels such that elements in the coarsest mesh is nearly the shape of a cube. In the numerical experiments this usually corresponds to a number among 4, 5 or 6.

![Fig. 4 Multigrid levels from coarse to fine](image1)

![Fig. 5 Schematic diagram of V-cycle with various operations](image2)

| Criteria         | MGCG      | Direct solver |
|------------------|-----------|---------------|
| Memory           | $N$       | $N\sqrt{N}$  |
| Computational time| $N$       | $N\sqrt{N}$  |
| Precision        | Approximate| Precise      |
4 GPU implementation of SIMP method

MG techniques, although relatively efficient, also suffers from the curse-of-dimensionality; this is particularly true when dealing with real-life systems having millions of degree-of-freedom. In this section, we propose a hybrid implementation of the SIMP method that exploits both GPU and MPI programming. We first discuss the CUDA architecture for GPU implementation followed by the proposed framework.

Algorithm 2: Two-grid algorithm $u = MG(u, f, K, h, S)$

1. Pre-smooth: $u^h = u^h + S^{-1}(f^h - K^h u^h)$ on $\Omega^h$
2. Residual: Compute Residual $r^h = f^h - K^h u^h$
3. Restriction: Projection of $r^h$ to $r^{2h}$ to $\Omega^{2h}$. $r^{2h} = I_h^{2h} r^h$
4. Solution on coarse Grid: solve on $\Omega^{2h}$. $K^{2h} e^{2h} = r^{2h}$
5. Interpolation: Projection of $e^{2h}$ to $\Omega^h$. $e^h = I_h^{2h} e^{2h}$
6. Update: $u^h = u^h + e^h$
7. Post-smooth: $u^h = u^h + S^{-1}(f^h - K^h u^h)$ on $\Omega^h$
8. return: $u$

Algorithm 3: Multigrid Method with V-cycle $u^h \leftarrow MG_v(u^h, f^h, S, h)$ of equation $K^h u^h = f^h$

1. Pre-smoothing: $u^h = u^h + S^{-1}(f^h - K^h u^h)$ on $\Omega^h$
2. if $\Omega^h$ is the coarsest grid then
   3. Solve the problem directly
3. else
   4. Restrict to next coarser grid: $r^{2h} = I_h^{2h} (f^h - K^h u^h)$
   5. Set initial iterate on next coarser grid: $u^{2h} = 0$
   6. Call the V-cycle scheme one time for next coarser grid: $u^{2h} \leftarrow MG_v(u^{2h}, f^{2h})$
4. end
5. Prolongation correction: $u^h = u^h + I_{2h} u^{2h}$
6. Post smoothing: $u^h = u^h + S^{-1}(f^h - K^h u^h)$ on $\Omega^h$
7. Return: $u$
4.1 CUDA architecture

GPUs were created to meet the market's need of fast and realistic 3D rendering in real time. Their tremendous computational capability at a reasonable cost is making them increasingly attractive in non-graphics HPC applications. The schematic diagram of a modern Turing architecture GPU is depicted in Fig. 6a, b below. Compute Unified Device Architecture (CUDA), a programming paradigm developed by Nvidia, is currently the most widely used GPU programming model. So-called data-parallel computing (data/SIMD parallelism) can be performed on the GPU by leveraging several processor cores. “Kernel”, a C Language Extension function, is used to run the parallel code (one instruction, multiple data). According to, the kernel call should indicate the number of CUDA threads structured as a grid of thread blocks, as seen in Fig. 6a [41]. The software level hierarchy is depicted in Fig. 6c [41].

The CUDA Architecture consists of numerous components (Fig. 6), including (i) NVIDIA GPU’s Parallel computation Engines (computation blocks), (ii) OS hardware initialization support, (iii) Kernel-level support, (iv) User mode driver providing developers with a device level API, (v) Set of parallel computing functions and functions via PTX instruction architecture (ISA). We exploit first four feature in the implementation strategy described below.

4.2 Hybrid topology optimization framework

We revisit the standard topology optimization algorithm using SIMP method (Algorithm 4). We note that step 6 corresponds to majority of the computational cost. One obvious solution is to leverage the powerful CUDA platform. However, CUDA has limited memory and hence, shifting the computation to CUDA will compromise the scalability of the algorithm. To address this issue, we propose a computing framework that leverage the strength of both GPU and Open Multi-Processing using CPU cores.

---

**Algorithm 4: Topology Optimization Framework**

1. **Input data**: design domain geometry and discretization (nelx, nely, nelz), loading \( f \) and boundary condition (fixeddofs), target volume fraction
2. **Initialize**: Initialization of empirical parameters \( \rho, p, r, m, \eta, ch = 1, k = 0 \)
3. **Stiffness Matrix**: Compute \( K_0 \)
4. **Prepare filter**: Compute \( w \) \( \triangleright \) Eq. 4, 5
5. **while** \( ch > 0.01 \) **do**
6. **Compute modified stiffness** \( K_e \) \( \triangleright \) Eq. 2
7. **Direct solve for** \( u \) **in** \( K_e u = f \)
8. **Sensitivity computation** \( \frac{\partial c}{\partial \rho} \) \( \triangleright \) Eq. 7
9. **Filtering of sensitivities** \( \triangleright \) Eq. 3
10. **Update** \( \rho_{k+1} \) **from** \( \rho_k \) \( \triangleright \) Eq. 8, 9
11. \( k = k + 1 \) \( \triangleright \) loop count
12. **end**
13. **Return**: The final \( \rho \) and visualization
The proposed topology optimization framework has multiple components. First and foremost, we replace the direct solver in step 6 of Algorithm 4 with Multigrid preconditioned conjugate gradient (MGCG) [42]. To further optimize the process, the MGCG solver’s computation has been offloaded to GPU’s highly efficient computational architecture. A 3-Dimensional MATLAB code for MGCG-based minimal compliance topology optimization has been developed here using the density based SIMP formulation. The computational intensive linear equation solver part in redesign loop of topology optimization is written in CUDA C language and compiled using NVIDIA nvcc compiler and called from MATLAB backbone using it’s mex function capability. It is worthwhile to mention that the program developed utilizes both GPU and CPU cores. To be specific, the computationally expensive operations like matrix multiplication and matrix inversions are carried out using GPU. For clarity of the readers, the broad framework of the proposed approach is shown in Fig. 7. The beauty of the developed framework is that the mex
function call can be used as a black box to solve any linear system of equations of the form \( Ku = f \) in cartesian discretization with 8-noded brick elements. Subsequently, it can be trivially incorporated within other topology optimization algorithms as well.

Next, we shift our focus to the MGCG algorithm within the proposed algorithm. Although MGCG is significantly efficient as compared to a direct solver, it still accounts for majority of the computational cost. To accelerate this step, we divide it into two parts as shown in Fig. 8; the first one corresponds to the conjugate gradient loop and the second one corresponds to the multigrid preconditioning inside the conjugate gradient loop. Preconditioning enhances the convergence rate and multigrid based preconditioning improves overall stability of the algorithm. The MATLAB code calls the compiled MGCG CUDA code using the mex function capability. Also, the main loop of MGCG algorithm is controlled on CPU memory. But the important variables are synchronized to GPU memory as well because the matrix multiplications and additions in steps 2 to 9 of MGCG algorithm in Fig. 8 are to be carried out in GPU in an heavily multithreaded environment. The addition and subtraction operation of step 3 and 4 can be carried out in CPU only as these don’t require huge multithreading. But flow of data from GPU device memory to host CPU memory adds time to overall process making it little less efficient. Hence these are also carried out in GPU. Similarly the main loop of multigrid V-cycle (which is basically step 6 of MGCG main

---

**Fig. 7** Topology optimization framework

**Fig. 8** CUDA (MGCG) framework
loop) is controlled in CPU which then subsequently calls GPU kernel functions for the operations in steps 1 to 7 etc.

Although GPU multithreading is highly recommended, sometimes unavailability of NVIDIA GPU or incompatibility issue prevent from execution of task. Hence two separate functions are developed, one is TopOptMCGOMP which is a purely CPU multi-threaded version (with no GPU related hardware and software requirement), and it utilizes open-source Open Multi-Processing(OpenMP) package to implement efficient CPU multi-threading. Another one is the hybrid TopOptCUDA version, which uses NVIDIA GPU using CUDA development package and CPU. Efficiency of both versions depend upon the relative strength of the CPU and GPU pair and numerical comparison between both the parts are given in Sect. 5. The challenge in the CUDA version is that although GPU’s have tremendous multithreading potential it’s native memory storage is quite limited and if it has to access data from host CPU memory, a significant time is spent on data transfer from CPU to GPU readable memory. Hence effective strategy has been developed to store as little data as possible in GPU memory without hindering the computation itself. This is described in Sect. 4.2.1. Also a simple homogenization technique is shown to facilitate further significant reduction in storage requirement in Sect. 4.2.2.

4.2.1 Use of local stiffness matrix for various computation

Instead of using assembled stiffness matrix for each and every computation, majority of the computation has been done with local stiffness matrices. This essentially saves precious GPU memory used for extremely fast computation. Each node inside of design domain is common to 8 elements, so sum of density times local stiffness matrices of these 8 elements gives the value equal to global stiffness value of that node. This strategy helps to perform nodal computations of all matrix and vector product and addition without explicit storage of global assembled matrices. In previous two subsections while calculating residue for each iteration in MGCG similar strategy can be seen to be in use. Although this increases number of floating point operation by roughly two times this technique decreases memory consumption significantly.

4.2.2 Homogenization strategy

To optimize the memory requirement, we utilize the well-known homogenization strategy. This is a simple yet effective technique to further optimize memory requirements. Traditionally while moving from a fine grid for the equation to coarse grid in V-cycle, restriction operation is used on \( f \) and \( K \) for transformation \( f^{2h} = I_{h}^{2h}f^{h} \) as shown in Fig. 3. The restriction operation is generally a Galerkin approximation given by equation:

\[
I_{h}^{2h} = \frac{1}{4} \begin{bmatrix}
1 & 2 & 1 \\
1 & 2 & 1 \\
& & \\
& & \\
& & \\
& & \\
& & \\
& & \\
\end{bmatrix},
\]

and,

\[
K^{2h} = I_{h}^{2h}K^{h}I_{h}^{2h}.
\]

This type of restriction leads to storage of local element stiffness matrices at coarser level. On one hand this eliminates the need for on the fly computation; however, the memory requirement increases. To reduce the memory requirement, we employ a homogenization scheme where the density of the coarse element is computed as mean of density values of eight neighboring elements at the finer level; this enables on the fly computation of the local stiffness matrix and reduces the need for storage.

4.2.3 Memory

Suppose a structural design domain has \( n \) degrees of freedom. Referring to MGCG algorithm in Fig. 8, the memory cost of the GPU can be divided into two parts; one corresponding to storage requirement in the preconditioned conjugate gradient (PCG) loop and the other part is in the V-cycle. There are four unique variables \((P, Q(=AP), R, Z)\) in the PCG iterations and five others in the V-cycle iterations as shown in Table 2. U denotes the current solution approximation, \( F \) is the vector of the right hand side. \( R \) is the vector of the defect(residue) and \( CX (=E\rho) \) is the vector of elements’ material properties (used for calculating \( K_e = CX \times (K_e)^h \)) which have dimension equals to number of elements (for big number of elements we can take it as approximately of size of \( \text{ndof} \)). AD is the vector of assembled main diagonal of stiffness matrix and is used in relax step of the V-cycle. Those vectors have dimension ndof on finest level, ndof/8 on coarser level, ndof/64 on next and

| Loop   | Vectors | Dimension of each vector | Sub total |
|--------|---------|--------------------------|-----------|
| PCG    | \( P, Q, R, Z \) | \( n \times 1 \) | \( 4n \) |
| V-cycle | \( U, F, R, CX, AD \) | \( n \times 1 \) | \( 5n \) |
| Total  |                      | \( 4n + 1.2 \times 5n = 10n \) |           |

\( ^1 \) A is computed on the fly and hence only \( Q \) is actually stored.
Each of these variables are of dimension $n \times 1$. So the total storage comes out to be $4n + 5n = 9n$. But in the V-cycle loop, the variables are also required to be stored in each successive coarser levels each having 1/8 times the size of previous finer level. Thus, the storage requirement due to the V-cycle part can be increased by 20% and hence, the total storage requirement is $4n + 1.2 \times 5n = 10n$.

For clarity of readers, the above calculation is summarized in Table 2.

To further illustrate the memory requirement, we consider a structural system having 100 millions degrees of freedom. We further assume that each variable has double precision. With this setup, the storage requirement will be approximately equal to $100 \times 10^6 \times 10 \times 8$ bytes = 8 GB. Decreasing the precision will even increase the capability of solving bigger size problems. Even with double precision it is quite modest considering 100 million elements can be fit into a standard desktop GPU having 8 GB of memory these days.

### 4.3 Parallelization strategy

The core operations of the MGCG method involves several matrix–vector and matrix–matrix multiplications which are tedious in large scale problems. The main idea is to write kernel functions to speed it up. This essentially allocates several element-wise operations of the multiplications to the massive threads of the GPU parallel computation engine. Three dimensional thread blocks are called for massive parallelization. One example for the function Defect used for computation of the residue $R = F - KU$ is shown in Figs. 9 and 10.

![Code snippet for sample kernel call](image)

Fig. 9 Code snippet for sample kernel call

CUDA Api provides 2 or 3 dim grid of blocks (typical maximum of each grid dimension 65 K). Each block could execute many threads (typical maximal number of threads per block 1024). Dimensions are nominal values and do not correspond to actual number of cores. In code above block grid dimensions are chosen so computation for each triad $i, j, k$ from cycle of residual computation from previous slide is performed by 1 thread. Variables blockIdx.x, blockIdx.y, threadIdx.x are provided by CUDA API. They identify current block and thread and take values: $\text{blockIdx.x} = 0, \ldots, nx/2 - 1$, $\text{blockIdx.y} = 0, \ldots, ny/2 - 1$, $\text{threadIdx.x} = 0, \ldots, nz/2 - 1$. Thus, computations are performed for all elements. Again we use external cycle over $k0, j0, i0$ to provide absence of interference between threads.

### 5 Numerical experiments

In this section, we present four numerical examples to illustrate the efficacy and robustness of the proposed approach. The examples are arranged in the increasing order of complexity and involves real-life scenarios such as bridges, buildings, and design dependent loading. We compare the results obtained with those obtained using the state-of-the-art topology optimization algorithms currently available (TOP3D125 and GPU based topology optimization code [43]). Values of some of the standard parameters used are: Young modulus of solid = $E_0 = 1$, Young modulus of void $E_{\text{min}} = 1e^{-9}$, Poisson ratio $\nu = 0.3$, penalty power $p = 3$, MGCG tolerance $= 10^{-8}$.

Computational time, no. of iterations, and average memory required have been considered as comparison metrics. Finally, to illustrate the versatility of the proposed approach, the developed topology optimization framework is tested on three computational environment: (a) standard GPU workstation with 8GB VRAM (System 1), (b) old GPU workstation with 2GB VRAM (System 2), and (c) regular GPU laptop with 4GB VRAM (System 3). The detailed specifications of the computational environmental is described in the appendix.
5.1 Example 1: 3D cantilever beam

As the first example, we consider a 3D cantilever beam subjected to line load $q$. The loading, boundary condition and design domain are shown in Fig. 11. As already stated, the objective here is to minimize the structural compliance. We consider two separate cases, one where the design domain is discretized into $64 \times 32 \times 32$ elements and another where the design domain is discretized into $128 \times 64 \times 64$ elements. Subsequently, the two cases have 65,536 and 524,288 design variables.

Figure 11b, c show the optimal topology obtained using the method proposed in [25] and the proposed approach for grid size of $64 \times 32 \times 32$. We observe that the results obtained using the proposed approach matches exactly with that obtained using [25]; this essentially validates the accuracy of the proposed approach. For illustrating the efficiency of the proposed approach, we compare the computationally efficiency and memory requirements. Table 3 depicts the comparison of the proposed approach with the state-of-the-art topology optimization framework proposed in [25]. Additionally we show the computational gain achieved by including only multigrid preconditioned iterative solver (TOP3D125MGCG), multigrid preconditioned iterative solver parallelized using OMP (TOP3D125MGCGOMP), and multigrid preconditioned iterative solver parallelized using GPU and CUDA (TOP3D125MGCGCUDA). We observe that TOP3D125MGCGCUDA on the standard workstation is approximately 45 times faster as compared to the optimized code presented in [25]. As compared the OMP version of the code, the CUDA version proposed is about 3 times faster. The gain in computational memory is even more significant with the proposed approach. While the TOP3D125 [25] requires 15.08 GB memory, the proposed TOP3D125MGCGOMP and TOP3D125MGCGCUDA require only 3.27, 2.60, and 2.03 GB memory only. It is noteworthy to mention that the memory for TOP3D-MGCGCUDA represents GPU memory. The advantage of the proposed framework becomes more prominent for grid size of $128 \times 64 \times 64$. The state-of-art Top125 fails in this case due to huge memory requirement. The MGCG based frameworks, one the other hand, yields satisfactory results with minimal increase in the memory requirements. The CUDA based framework is approximately 2 times and 1.5 times faster as compared to TOP3D125MGCG and TOP3D125MGCGOMP, respectively. Similar observations can be found when run on systems 2 and 3 respectively (Fig. 12).

Finally, to illustrate the scalability of the proposed GPU based framework, we conduct a case study by varying the number of degrees of freedom of the system (achieved by varying the discretization). We compare TOP3D125MGCG and TOP125MGCGCUDA. It is observed that both memory and time per iteration required for the proposed CUDA based topology optimization framework increases at a much slower rate as compared to the MGCG version. This indicates the superior scalability of the proposed approach.

One of the important component of the proposed approach is the homogenization scheme as it helps in addressing the bottleneck associated with GPU memory. To illustrate the advantage gained by using the homogenization scheme, a comparative study with and without homogenization is presented in Table 4. We observe that the use of homogenization reduces the memory requirement by 40–50% depending on the problem size and other factors (Fig. 13).

5.2 Example 2: arch bridge

As the second example, we illustrate the performance of the developed framework on passive void and passive solid region within design domain to obtain bridge like shapes similar to [44]. Similar to previous example, we study two different cases. In the first case, we consider a $140 \times 10 \times 20$ m design domain. The top layer having 1.5 m thickness is considered as passive solid region (non-design domain) as shown in Fig. 14a. Similarly, at the midway of 140 m length...
Fig. 12  a Cantilever Beam (64 × 32 × 32). Convergence of objective function b Time of iterations in system 1. c Time of iterations in system 2. d Comparison of CUDA codes in 3 systems

### Table 3 Comparison of performance of various codes in system 1

| Code           | Discretization | Time per iteration | Memory requirement (GB) | No of iterations to converge | Compliance (N-m)     |
|----------------|----------------|--------------------|-------------------------|-----------------------------|----------------------|
| TOP3D125       | 64 × 32 × 32   | 44.7 s             | 15.08                   | 64                          | 7.35 × 10^3          |
| TOP3D125MGCG   | 64 × 32 × 32   | 6.4 s              | 3.27                    | 64                          | 7.35 × 10^3          |
| TOP3D125MGCGOMP| 64 × 32 × 32   | 2.6 s              | 2.60                    | 64                          | 7.35 × 10^3          |
| TOP3D125MGCGCUDA| 64 × 32 × 32  | 1.0 s              | 0.04 (2.03)             | 64                          | 7.35 × 10^3          |
| TOP3D125      | 1128 × 64 × 64 | Out of memory      |                         |                             | 9.73 × 10^3          |
| TOP3D125MGCG  | 128 × 64 × 64  | 44.9               | 3.98                    | 49                          | 9.73 × 10^3          |
| TOP3D125MGCGOMP| 1128 × 64 × 64| 33.8 s             | 2.35                    | 49                          | 9.73 × 10^3          |
| TOP3D125MGCGCUDA| 1128 × 64 × 64| 21.3 s             | 0.414 (2.79)            | 49                          | 9.73 × 10^3          |
a small void region of thickness 1 m is treated as passive void. Boundary conditions and loading include simple support at four bottom corners and an uniformly distributed load (UDL) of 100 N/m² on the top surface.

A discretization of $448 \times 32 \times 64$ is considered with a target volume fraction of 0.14. The resultant shape of compliance minimization problem is shown in Fig. 14c. The output is an arch bridge whereas an easy guess could be that of a bench kind of shape. Compared to the output from [44], our
result looks mostly similar; this validates the accuracy of the proposed approach for this problem.

A second similar case is shown in Fig. 15 where the span of the design domain is 40 m. Carriageway width and height are respectively 10 m and 20.6 m. At the mid-height, a non-designable (passive solid) layer of 0.6 m thick is considered corresponding to the deck of the bridge. Also just above it, a void region of 40 m × 8.8 m × 10 m is considered corresponding to space required for vehicular passage. The uniformly distributed loading on the deck and the boundary conditions are shown in Fig. 15a. The optimization is carried out with using a discretization of 256 × 64 × 128 which accumulates to around 2 millions 8-noded 3D elements. The final shape after 49 iterations comes out to be an arch bridge as shown in Fig. 15b.

| Design domain dimension | Discretization | Total no. of iterations | Time per iteration (s) | Average memory (GB) | Compliance (Nm) |
|-------------------------|----------------|-------------------------|------------------------|---------------------|-----------------|
| 140 m × 10 m × 20 m     | 448 × 32 × 64  | 57                      | 26.27                  | 0.486               | 9.4 × 10⁸       |
| 40 m × 10 m × 20 m      | 256 × 64 × 128 | 49                      | 224.18                 | 1.14                | 4.5 × 10⁸       |
The summary of computing time and memory requirement of two type of arch bridges described above is summarized in Table 5. We observe that the proposed hybrid topology optimization framework is highly efficient both in terms of computational efficiency (26.27 and 224.18 s per iteration) and computational memory (0.486 GB and 1.14 GB) and can run even on an old system with 2 GB GPU memory.

5.3 Example 3: high rise building

In this example, a high rise tall building has been analyzed. The building is acted upon by the wind loading and the lateral load resisting frame system is to be obtained from minimization of compliance. The plan dimension of the building is 64 m × 64 m. The height to plan dimension ratio $H/B = 4$. For simplicity, the base of the building is kept fixed and a constant magnitude of lateral loading is applied at all floor levels across the elevation of the building on one side of the building. A similar work is recently reported in [45] in which the authors have obtained a diagrid pattern (Fig. 16c) of eccentric bracing system by aligning material along principal stress directions.

In our experiment, a discretization of $64\times 64 \times 256$ is considered. Subsequently, we have over 1 million design variables. The core of the building is kept hollow. A bracing system similar to [45] has been obtained corresponding to a volume fraction of 0.12 in the active perimeter region. As for computational time, the proposed approach takes around 33 s per iteration and 64 iteration to yield

---

2 For numerical stability, we consider a very small stiffness of $1/10^6$. 

---

Fig. 16 High rise building: a Problem description [45]. b Concept design of real world transit bay tower [46]. c Various principal stress inspired design from [45]. d Our 2D designs. e Our 3D design. f 3D slender design
The final optimized configuration is shown in Fig. 16. The converged compliance value is \(5.91 \times 10^5\) Nm. The proposed approach require only 0.71 GB of GPU memory. The converged solutions. As for computational memory, the proposed approach require only 0.71 GB of GPU memory. The converged compliance value is \(5.91 \times 10^5\) Nm. The final optimized configuration is shown in Fig. 16.

5.4 Example 4: foot bridge

As the fourth example, we optimize a foot bridge for gravity loading. Since gravity load is dependent on the amount of material present, the load varies with each iteration and complicates the optimization process. Subsequently, higher number of iterations is necessary to achieve converged solution. The design domain for the problem is shown in Fig. 17a wherein gray parallelepiped whose surface is the passive void region and internal light green tube is passive solid region. Dark blue layer represents the active region. Gravity loading proportional to volume of active and passive solid is applied in each optimization iteration and support structure for the tubular passage is obtained. A discretization of \(1152 \times 64 \times 256\) (which corresponds to approximately 18.9 millions of elements and 58 millions of degrees of freedom) is considered and optimization is carried out for a target volume fraction of 0.125 in the support structure region. The optimized configuration obtained using the proposed approach is shown in Fig. 17b. We observe that the pattern is almost similar to an actual solution from literature [44]. Overall the proposed approach converges in 49 iterations and requires 3.924 GB GPU memory as detailed in Table 6.

5.5 Example 5: topology optimization of high rise building

As the fifth example, we consider topology design of a high rise building having plan area of \(54\) m \(\times\) \(54\) m. The height of the building is 162 m. The objective here is to illustrate the capability of the proposed algorithm in solving a highly heterogeneous system. We have considered the system to be subjected to parabolic loading profile [45]. We consider the floors to be of concrete with Elastic modulus of 25,000 MPa. The objective here is to minimize the structural compliance; however, unlike the previous examples, we keep a track on the maximum top story drift following the Eurocode for building [47, 48]. The analysis of this structure is carried to calculate displacement subjected to the lateral loading.

Unlike the problems presented till now, this problem exposes the bottleneck associated with limited GPU memory, and it was not possible to perform topology optimization without employing the homogenization scheme discussed before. We utilize the proposed homogenization scheme and represent the density of an element by computing the average value of eight finer elements; this reduces the storage requirement of stiffness matrices of each element in the coarser grids in the V-cycle part of the algorithm. Table 7 summarizes the performance of standard Galerkin scheme and homogenization based scheme on this system. We observe that the homogenization approach has around 43% less cost to GPU memory and its iterations takes less time due to faster convergence of the MGCG iterations per each optimization redesign loop. The optimized topology obtained is shown in Fig. 18. We observe that the top floor drift for the optimized configuration is 0.026 m which is significantly lower than the allowed threshold.

5.6 Example 6: conceptual design of Qatar National Convention Center

In this example, we have considered the real life structure of Qatar National Convention Center. It was completed by Arata Isozaki in 2011 and it is inspired from shape of a tree. Because of unavailability of complete design details, we considered a simpler design domain of \(480\) m \(\times\) \(20\) m \(\times\) \(80\) m. A uniformly distributed loading is applied in the gravity direction and one fourth of the structure is analyzed considering the symmetry in two directions. The problem domain is discretized into \(1536 \times 64 \times 256\) elements; this corresponds to 25.5 millions design variables. The optimized topology is shown in Fig. 19 and corresponding iterations and memory requirement is summarized in Table 8. Although the overall shape is coming somewhat similar, the actual existing shape is seen to be different than that of the shape obtained from our code. This can be attributed to the fact that the topology optimization gives a structural form for conceptual design whereas the final existing shape is the outcome of the subsequent rigorous detailed design phase where many member sizes are modified to satisfy the building codal safety provisions. Moreover the detailed loading, support conditions, constraints and all detailed dimensions are not available in the open literature. Nonetheless, the optimal topology obtained using our framework broadly resembles the existing structure.

5.7 Scalability and efficiency of the proposed approach

In this section, we examine the efficiency and scalability of the proposed approach. To illustrate the efficiency of the proposed framework, we have compared computational time required using only CPU with parallel computing, only GPU
with parallel computing and our proposed framework. We have considered the same problem presented in Sect. 5.6 with a discretization of $1024 \times 48 \times 128$. It is tested on a workstation with a fairly powerful CPU (Intel Xeon Gold 5218) and GPU (Nvidia Quadro RTX 4000). The results have been shown in Fig. 20. It is observed that the proposed hybrid version is most efficient followed by only GPU implementation and only CPU implementation, respectively.

The scalability of our parallel algorithm has been tested in a cantilever beam described in Sect. 5.1. Both strong and weak scaling has been tested. First three different GPUs are tested with the same problem size for strong scaling. Only time duration of the parallelized part executed by our framework are compared. For examining strong scaling, the overall problem domain is discretized into $128 \times 64 \times 64$ elements. Three different GPUs with increasing order of computational power are compared, and corresponding time and speedups are summarized in Table 9. It is observed that the efficiencies are coming close to unity indicating very good scaling for parallel execution with increase in cuda

![Design domain highlight. Optimized support system for the foot bridge obtained. Solution from literature [44]](image)

Table 6  Foot bridge analysis details

| Design domain dimension | Discretization | Total no. of iterations | Time per iteration | Avg. memory | Compliance |
|-------------------------|----------------|-------------------------|--------------------|-------------|------------|
| 180 m x 10 m x 40 m     | 1152 x 64 x 256| 49                      | 2850 s             | 3.924 GB    | 6.67 x 10^{19} Nm |

![Optimized configuration for the high-rise building problem](image)
cores. The reported efficiency has been calculated using the following definition:

$$\text{efficiency} = \frac{\text{speedup}}{\text{normalized computational power}}$$  \hspace{1cm} (24)

To examine the weak-scaling of the developed framework, we increase the number of design variables by changing the discretization. For this case also, computational time and efficiencies are compared. The efficiency has been calculated as follows:

$$\text{efficiency} = \frac{\text{present computational time}/\text{base computational time}}{\text{normalized problem size}/\text{normalized computational power}}$$  \hspace{1cm} (25)

It is observed that the efficiency results are within reasonable range as summarized in Table 10. It is worthy to note that the number of conjugate gradient iteration required changes with change in discretization and can potentially alter the overall efficiency. For example, the number of conjugate gradient iterations were higher for $128 \times 96 \times 96$ and this is reflected in the efficiency column.

| Approach            | Discretization | Total no. of iterations | Time per iteration (s) | Avg. memory (GB) |
|---------------------|----------------|-------------------------|------------------------|------------------|
| Standard (Galerkin) | $64 \times 64 \times 320$ | 84                      | 102.58                 | 1.08             |
| Homogenization      | $64 \times 64 \times 320$ | 81                      | 86.37                  | 0.62             |

Table 8 QNC analysis details

| Design domain dimension | Discretization | Total no. of iterations | Time per iteration | Avg. memory | Compliance |
|-------------------------|----------------|-------------------------|--------------------|-------------|------------|
| $480 \text{ m} \times 20 \text{ m} \times 80 \text{ m}$ | $1536 \times 64 \times 256$ | 50                      | 2554 s            | 5.4 GB      | $9.1 \times 10^{13} \text{ Nm}$ |

6 Conclusion

In this work, we have proposed an efficient implementation of a new GPU enhanced hybrid topology optimization framework for structural compliance minimization. The proposed approach replaces the primary bottleneck associated with direct solution of state space system with an efficient multigrid conjugate gradient method. The present novel approach builds upon the framework of [25] as driving topology optimization algorithm along with replacement of the finite-element solution module with our custom module called from MATLAB with its MEX interface which utilizes both CPU and GPU. In specific, all tedious arithmetic computations have been shifted to the highly multithreaded cores of modern GPU based on CUDA architecture. In this novel implementation strategy, we have utilized a simple homogenization scheme within the proposed approach that drastically reduces the memory requirement and improves the computational efficiency. The main novelty lies in the strategic implementation of the SIMP – Multigrid Preconditioned Conjugate Gradient – (Matrix on the fly + Homogenization) – Cuda functionality which makes the proposed approach highly efficient and easily scalable to systems having millions of degree of freedoms. The strong and weak scaling analyses of the proposed framework show extremely good efficiency.

Two version of our framework has been developed and implemented—one for CUDA GPU based system (TOP-3D125CUDA) and another for purely CPU system (TOP-3D125OMP). Several examples are solved to illustrate the
performance of the proposed approach. In a standard cantilever beam benchmark problem, our CUDA based algorithm is about two times faster and consumes 7–8 times lower memory compared to contemporary efficient implementation. Similarly the OMP version is also 1.5 times faster, although it consumes similar memory compared to the state-of-the-art implementations. It is worthwhile to note that the increased efficiency is achieved without any compromise in the accuracy. One of the primary feature of the proposed approach is its scalability. It easily scales to millions of degrees of freedom. The proposed framework consumes only 1.1 GB of GPU memory and computational time of around 4 min for solving the arch-bridge problem having over 2 million degrees of freedom. A real-life standing structure of Qatar National Convention centre has also been simulated. A further extension of our algorithm to incorporate a homogenization scheme is shown to reduce the memory requirement by around 40–50% in the cantilever benchmark problem. Overall, the proposed approach is accurate, efficient, scalable, and can easily be integrated with any existing topology optimization algorithm. Faster convergence of the homogenization approach in the high rise building having high heterogeneity has also been demonstrated. Overall our framework produced excellent results across various examples with significant efficiency in computation and memory requirement.

Despite the excellent performance, the proposed framework has certain limitations. Firstly, in its current form, the proposed framework is applicable to structured mesh. For unstructured mesh, an additional step involving dividing the problem domain into sub-regions having similar discretization is necessary. Secondly, the developed CUDA based framework will only work with NVIDIA GPUs. This is not a significant limitation when one considers the fact that NVIDIA currently has over 77% [49] market share. Also, the developed OMP version of the framework is universal and will work on any windows based system. Thirdly, we have only focussed on compliance minimization. The proposed framework can be extended to solve stress based and multi-physics topology optimization problems In future, some of these limitations will be addressed.

Table 9 Strong scaling analysis details

| GPU computational power (TFLOPs) | Normalized computational power | Time (s) | Speedup | Efficiency |
|----------------------------------|--------------------------------|----------|---------|------------|
| 2.98 (Nvidia GTX 1650 mobile)    | 1                              | 18.23    | 1       | 1.00       |
| 3.82 (Nvidia Quadro P2200)      | 1.28                           | 14.31    | 1.27    | 0.99       |
| 7.12 (Nvidia Quadro RTX4000)    | 2.39                           | 7.56     | 2.41    | 1.00       |

Table 10 Weak scaling analysis details

| GPU computational power (TFLOPs) | Normalized computational power | Normalized problem size (discretization) | Time (s) | Efficiency |
|----------------------------------|--------------------------------|------------------------------------------|----------|------------|
| 2.98 (Nvidia GTX 1650 mobile)    | 1                              | 1.00 (128 × 64 × 64)                     | 18.23    | 1.00       |
| 3.82 (Nvidia Quadro P2200)      | 1.28                           | 1.25 (128 × 80 × 64)                     | 18.07    | 1.03       |
| 7.12 (Nvidia Quadro RTX4000)    | 2.39                           | 2.25 (128 × 96 × 96)                     | 24.80    | 0.78       |
Appendix 1: Three different system configurations used for testing

See Table 11.

Table 11 Specification of systems used for testing

| Component   | System-1          | System-2          | System-3          |
|-------------|-------------------|-------------------|-------------------|
| CPU         | 2 × Intel Xeon Gold 5218 | 2 × Intel Xeon X5660 | Intel i7 9750H    |
| GPU         | NVIDIA QUADRO RTX4000 with 8GB GDDR6X VRAM | NVIDIA QUADRO K620 with 2GB GDDR3 VRAM | NVIDIA GeForce GTX 1650 with 4GB GDDR5 VRAM |
| RAM         | 192 GB DDR4       | 48 GB DDR3        | 16 GB DDR4        |
| OS          | Windows 10 x 64 20H2 | Windows 10 x 64 20H2 | Windows 10 x 64 20H2 |
| MATLAB      | R2020b            | R2020b            | R2020b            |
| CUDA toolkit | 10.2              | 10.2              | 10.2              |

Acknowledgements  APP acknowledge the financial support received from IIT Roorkee in form of MHRD, Govt. Of India, fellowship. SC acknowledges the financial support received from Science and Engineering Research Board via Grant no. SRG/2021/000467.

Declarations

Conflict of interest  The authors declare that they have no conflict of interest.

Replication of results  Matlab codes as well as the dependent CUDA C routines for the examples shown in this work are available at https://github.com/cscem-iitd/GPU-TO.

References

1. Bendsøe MP, Kikuchi N (1988) Generating optimal topologies in structural design using a homogenization method. Comput Methods Appl Mech Eng 71(2):197–224
2. Yoon GH, Choi H, Hur S (2018) Multiphysics topology optimization for piezoelectric acoustic focuser. Comput Methods Appl Mech Eng 332:600–623
3. Zhu X, Zhao C, Wang X, Zhou Y, Hu P, Ma Z-D (2019) Temperature-constrained topology optimization of thermo-mechanical coupled problems. Eng Optim
4. Kazemi H, Vaziri A, Norato JA (2020) Multi-material topology optimization of lattice structures using geometry projection. Comput Methods Appl Mech Eng 363:112895
5. Chakraborty S, Goswami S, Rabczuk T (2019) A surrogate assisted adaptive framework for robust topology optimization. Comput Methods Appl Mech Eng 346:63–84
6. Giraldo-Londoño O, Mirabella L, Dalloro L, Paulino GH (2020) Multi-material thermomechanical topology optimization with applications to additive manufacturing: design of main composite part and its support structure. Comput Methods Appl Mech Eng 363:112812
7. Liang X, To AC, Du J, Zhang YJ (2021) Topology optimization of phononic-like structures using experimental material interpolation model for additive manufactured lattice infills. Comput Methods Appl Mech Eng 377:113717
8. Sigmund O, Maute K (2013) Topology optimization approaches. Struct Multidiscip Optim 48(6):1031–1055
9. Kreissl S, Maute K (2012) Levelset based fluid topology optimization using the extended finite element method. Struct Multidiscip Optim 46(3):311–326
10. Wang Y, Wang Z, Xia Z, Poh LH (2018) Structural design optimization using isogeometric analysis: a comprehensive review. Comput Model Eng Sci 117(3):455–507
11. Gai Y, Zhu X, Zhang YJ, Hou W, Hu P (2020) Explicit isogeometric topology optimization based on moving morphable voids with closed B-spline boundary curves. Struct Multidiscip Optim 61(3):963–982
12. Sigmund O (2020) EML webinar overview: topology optimization-status and perspectives. Extreme Mech Lett 39:100855
13. Wadbro E, Berggren M (2009) Megapixel topology optimization on a graphics processing unit. SIAM Rev 51(4):707–721
14. Haase G, Liebmann M, Douglas CC, Plank G (2010) A parallel algebraic multigrid solver on graphics processing units. Springer, Berlin, pp 38–47
15. Gandham R, Esler K, Zhang Y (2014) A GPU accelerated aggregation algebraic multigrid method. Comput Math Appl 68(10):1151–1160
16. Schmidt S, Schulz V (2011) A 2589 line topology optimization code written for the graphics card. Comput Vis Sci 14(6):249–256
17. Martínez-Frutos J, Martínez-Castejón Pj, Herrero-Pérez D (2015) Fine-grained GPU implementation of assembly-free iterative solver for finite element problems. Comput Struct 157:9–18
18. Xia Z, Wang Y, Wang Q, Mei C (2017) GPU parallel strategy for parameterized LSM-based topology optimization using isogeometric analysis. Struct Multidiscip Optim 56(2):413–434
19. Dick C, Georgii J, Westermann R (2011) A real-time multigrid finite hexahedra method for elasticity simulation using CUDA. Simul Model Pract Theory 19(2):801–816
20. Martínez-Frutos J, Martínez-Castejón PJ, Herrero-Pérez D (2017) Efficient topology optimization using GPU computing with multi-level granularity. Adv Eng Softw 106:47–62
21. Martínez-Frutos J, Herrero-Pérez D (2017) GPU acceleration for evolutionary topology optimization of continuum structures using iso surfaces. Comput Struct 182:119–136
22. Baiges J, Martínez-Frutos J, Herrero-Pérez D, Otero F, Ferrer A (2019) Large-scale stochastic topology optimization using adaptive mesh refinement and coarsening through a two-level parallelization scheme. Comput Methods Appl Mech Eng 343:186–206
23. Li H, Li H, Gao L, Li J, Li P, Yang Y (2021) Topology optimization of arbitrary-shape multi-phase structure with structured meshes based on a virtual phase method. Comput Methods Appl Mech Eng 387:114138
24. Xie X, Wang S, Xu M, Jiang N, Wang Y (2020) A hierarchical spline based isogeometric topology optimization using moving morphable components. Comput Methods Appl Mech Eng 360:112696
25. Ferrari F, Sigmund O (2020) A new generation 99 line Matlab code for compliance topology optimization and its extension to 3D. Struct Multidiscip Optim 62(4):2211–2228
26. Bendsoe MP (1989) Optimal shape design as a material distribution problem. Struct Optim 1(4):193–202
27. Bendsoe MP, Sigmund O (2003) Topology optimization: theory, methods, and applications. Springer, Berlin
28. Mlejnek H (1992) Some aspects of the genesis of structures. Struct Optim 5(1):64–69
29. Zhou M, Rozvany G (1991) The COC algorithm, Part II: topological, geometrical and generalized shape optimization. Comput Methods Appl Mech Eng 89(1–3):309–336
30. Sigmund O, Petersson J (1998) Numerical instabilities in topology optimization: a survey on procedures dealing with checkerboards, mesh-dependencies and local minima. Struct Optim 16(1):68–75
31. Bourdin B (2001) Filters in topology optimization. Int J Numer Meth Eng 50(9):2143–2158
32. Zeng S, Lee PS (2019) Topology optimization of liquid-cooled microchannel heat sinks: an experimental and numerical study. Int J Heat Mass Transf 142:118401
33. Sigmund O (1998) Topology optimization in multiphysics problems. In: 7th AIAA/USAF/NASA/ISSMO symposium on multidisciplinary analysis and optimization, p 4905
34. Wilson RB A simplicial algorithm for concave programming. Ph. D. Dissertation, Graduate School of Business Administration
35. Svanberg K (1987) The method of moving asymptotes—a new method for structural optimization. Int J Numer Meth Eng 24(2):359–373
36. Bendsoe MP, Sigmund O (1995) Optimization of structural topology, shape, and material, vol 414. Springer, Berlin
37. Papadrakakis M, Stavroulakis G, Karatarakis A (2011) A new era in scientific computing: domain decomposition methods in hybrid CPU-GPU architectures. Comput Methods Appl Mech Eng 200(13–16):1490–1508
38. Davis TA (2006) Direct methods for sparse linear systems. SIAM, Philadelphia
39. Hestenes MR, Stiefel E et al (1952) Methods of conjugate gradients for solving linear systems, vol 49. NBS, Washington, DC
40. Vassilevski PS (2008) Multilevel block factorization preconditioners: matrix-based analysis and algorithms for solving finite element equations. Springer, Berlin
41. CUDA C++ Programming Guide. NVIDIA Corporation, (2021)
42. Amir O, Aage N, Lazarov BS (2014) On multigrid-CG for efficient topology optimization. Struct Multidiscip Optim 49(5):815–829
43. Wu J, Dick C, Westermann R (2015) A system for high-resolution topology optimization. IEEE Trans Vis Comput Gr 22(3):1195–1208
44. Xie YM, Zuo ZH, Huang X, Black T, Felicetti P (2014) Application of topological optimisation technology to bridge design. Struct Eng Int 24(2):185–191
45. Cascone F, Faiella D, Tomei V, Mele E (2021) Stress lines inspired structural patterns for tall buildings. Eng Struct 229:111546
46. Sarkisian M, Lee P, Long E, Shook D (2010) Organic and natural forms in building design. Struct Congress 2010:2840–2851
47. Standard B Eurocode 1: actions on structures
48. Standard B (2004) Eurocode 2: design of concrete structures-, Part 1-1: general rules and rules for buildings, p 230
49. Otterness N, Anderson JH (2020) Amd gpus as an alternative to nvidia for supporting real-time workloads. In: 32nd Euromicro conference on real-time systems (ECRTS 2020), Schloss Dagstuhl-Leibniz-Zentrum für Informatik

Publisher’s Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Springer Nature or its licensor (e.g., a society or other partner) holds exclusive rights to this article under a publishing agreement with the author(s) or other rightsholder(s); author self-archiving of the accepted manuscript version of this article is solely governed by the terms of such publishing agreement and applicable law.