Engineering with Computers, Volume 35, July 2019, pp 1059–1079
DOI:10.1007/s00366-018-0651-1

Multi-Physics Bi-directional Evolutionary Topology Optimization on GPU-architecture

David J. Munk, Timoleon Kipouros and Gareth A. Vio

Received: date / Accepted: date

Abstract Topology optimization has proven to be viable for use in the preliminary phases of real world design problems. Ultimately, the restricting factor is the computational expense since a multitude of designs need to be considered. This is especially imperative in such fields as aerospace, automotive and biomedical, where the problems involve multiple physical models, typically fluids and structures, requiring excessive computational calculations. One possible solution to this is to implement codes on massively parallel computer architectures, such as Graphics Processing Units (GPUs). The present work investigates the feasibility of a GPU-implemented Lattice Boltzmann method for multi-physics topology optimization for the first time. Noticeable differences between the GPU implementation and a Central Processing Unit (CPU) version of the code are observed and the challenges associated with finding feasible solutions in a computational efficient manner are discussed and solved here, for the first time on a multi-physics topology optimization problem. The main goal of this paper is to speed up the topology optimization process for multi-physics problems without restricting the design domain, or sacrificing considerable performance in the objectives. Examples are compared with both standard CPU and various levels of numerical precision GPU codes to better illustrate the advantages and disadvantages of this implementation. A structural and fluid objective topology optimization problem is solved to vary the dependence of the algorithm on the GPU, extending on the previous literature that has only considered structural objectives of non-design dependent load problems. The results of this work indicate some discrepancies between GPU and CPU implementations that have not been seen before in the literature and are imperative to the speed-up of multi-physics topology optimization algorithms using GPUs.

Keywords Lattice Boltzmann method · Graphics processing units · Real world applications

D. Munk
The University of Sydney
Tel.: +61-(0)2-9351-7136
Fax: +61-(0)2-9351-7060
E-mail: david.munk@sydney.edu.au

Published by Springer. This is the Author Accepted Manuscript issued with: Creative Commons Attribution Non-Commercial License (CC:BY:NC 4.0).
The final published version (version of record) is available online at DOI:10.1007/s00366-018-0651-1. Please refer to any applicable publisher terms of use.

1 Introduction

Over the past two decades topology optimization has rapidly matured to a point where it can be used in real world design applications with minimal limitations (Munk et al (2015)). However, one such limitation is the computational resources required for large scale problems (Deaton and Grandhi (2014)). For engineering problems, the design space being considered is large and the objective function typically involves multiple, complicated, physical phenomena. Therefore, this leads to computationally intensive problems. The aim of this paper is to determine the feasibility of using GPUs with multi-physics topology optimization algorithms for real world design problems. The increase in computational efficiency due to the GPU architecture and quality of the final solutions are compared with the same problem implemented on a CPU.

Recently, reductions in computational expense are achieved by increasing the level of parallelism, i.e. increasing the number of computational cores while maintaining the same clock frequency, in the code (Zegard and Paulino (2013)). This has meant the development and use of many-core processors, which are processors that have evolved to a high-level of parallelism, for such tasks. GPUs are a class of many-core processors. GPUs have a different design approach compared with CPUs. CPUs are a general purpose multi-core processor containing many high level instructions, whereas GPUs are many-core processors that have a faster and smaller set of instructions, but are capable of handling a large number of concurrent threads. Therefore, GPUs can be used to drastically speed up computationally intensive problems and reduce the overall computational expense.

Topology optimization, generally speaking, aims to evolve an initial design towards an optimum one with regards to minimizing a given objective under several constraints (Bendsøe and Sigmund (2003)). Several approaches have been developed to guide the evolution of the topology towards the optimum (Sigmund and Maute (2013); Deaton and Grandhi (2014); Munk et al (2015)). These approaches can be divided into two main fields: continuous and discrete. Continuous methods, such as the Solid Isotropic Material with Penalization (SIMP) (Bendsøe (1989); Rozvany et al (1992)), apply a relaxation on the design variables so that their values can be inside the entire range defined by $[0,1]$. Discrete methods, such as Evolutionary Structural Optimization (ESO) (Xie and Steven (1993)) and Level-Set (LS) (Osher and Sethian (1988)), do not relax the problem and hence restrict the design variables to the boundaries of the range $\{0,1\}$. While some effort using SIMP and LS methods have been solved with GPU architectures (Aissa et al (2014)), only one recent study exists with ESO methods (Martinez-Frutos and Herrero-Perez (2017)) and only with structural optimization.

Application of the SIMP method to large-scale problems, with millions of design variables, has proven to be computationally demanding and therefore requires a high level of parallelism (Aissa et al (2014)). As an example, the work of Mahdavi et al (2006) demonstrates a SIMP method for topology optimization with parallelization on a CPU. Further, Vemaganti and Lawrence (2005) look at three different parallel linear solvers for SIMP topology optimization showing speed-up and reduced effects of ill-conditioning in the finite element problems. However, GPUs as an alternative low-cost-high-performance system have also been tested for solving topology optimization problems with SIMP methods. Schmidt and Schulz (2011) use SIMP on structured meshes with a matrix-free conjugate gradient solver, showing that it is faster than when a CPU with 48 cores shared memory is used. Such a strategy was also employed by Suresh (2013) for solving of the system of equations of elasticity, achieving speedups of one order of magnitude. A GPU-implemented SIMP method with a preconditioned conjugate gradient solver applied to a 2D plate with a

heat source yielded a speed up of 20 times compared with a single CPU and 3 times against a multi-threaded CPU (Wadbro and Berggren (2009)). This study was limited to single-precision format, due to the lack of a native double-precision support for early GPUs, which meant that convergence of the solver was not ensured due to round-off errors. Recently, a SIMP approach for unstructured meshes was implemented on a GPU by Zegard and Paulino (2013), focusing on assembly of the stiffness matrix. Furthermore, Wu et al (2016) used the geometric multi-grid preconditioning for the GPU instance of preconditioned conjugate gradient solvers to perform a reduced number of Finite Element Analyses (FEA), and iterations per FEA, in the SIMP algorithm. This is achieved by reducing the tolerance of the iterative method to increase the GPU performance. However, by using this configuration the solution is likely to arrive at a local optimum, meaning a more sound solution might exist (Wu et al (2016)). This loss in accuracy was assumed by the authors for the sake of efficiency.

In topology optimization the LS method evolves the boundaries of the structure by minimizing a given objective. The boundary evolution is solved using a finite difference method, which acts on a reduced group of elements, making it suitable for efficient GPU processing (Micikevicius (2009)). The LS method was implemented on a GPU architecture by Herrero et al (2013). Later, an inverse homogenization problem was solved with a GPU implementation of a LS method by Challis et al (2014), targeting high resolution topology optimization. They recorded an increasing speed-up with problem size, reaching 13 times speed-up for 3D problems containing over 4 million design variables.

Meta-heuristic optimization methods, where no gradients are taken, can also be used to solve real world problems of high-dimensionality (Martins and Lambe (2013)). These methods often use nature inspired algorithms, which are ideal for intelligently harnessing the capacity of GPUs. One such method, which uses a Tabu Search algorithm, has been implemented on a GPU architecture (Tsotskas et al (2014)). They showed that for problems of high dimensionality, defined as 270 variables or more, the GPU-implemented version of the code outperforms the CPU version. Up to a 12% speed up was recorded. Later, the same authors developed a GPU-implemented Lattice Boltzmann Method (LBM) and applied the TS algorithm to a micro fluidic device (Tsotskas et al (2015)). They noted that the most computationally intensive part of the process was the simulation of the flow via LBM. Therefore, the TS algorithm was employed on a CPU, since relatively small speed-ups were achieved (Tsotskas et al (2014)), and the LBM was employed on a GPU. The TS combined with the GPU-LBM delivered results approximately 20 times faster compared to an earlier system that employed a CPU-based LBM code (D'Ammaro et al (2010)). Recently, Laniewski-Wollk and Rokicki (2016) developed a discrete adjoint formulation for a wide class of LBMs. They implement their LBM on a GPU architecture for channel flow to design a free-topology mixer and heat exchanger using the Method of Moving Asymptotes (MMA) and a simple descent algorithm, separately. While the method was not compared to a CPU implementation of the code, it was shown to be efficient by demonstrating that the code had nearly linear weak scaling.

The limited literature on GPU-implemented topology optimization shows that the solver is the most time consuming part of the optimization and hence should be the focus during the adaptation to GPU architecture (Aissa et al (2014)). Furthermore the other procedures, such as the optimizer, are not computationally expensive and therefore not directly relevant for good acceleration through GPU. Hence, studies have focused on implementing the solver on GPU architectures without topology optimization. For purely structural topology optimization a finite element solver is used to determine the displacements of the structure under a given load. Cecka et al (2011) presented a GPU accelerated FEA code using an unstructured mesh, achieving a speed-up of 30 or more compared to an optimized double-

precision single core implementation. For a review of the literature on the use of GPUs in FEA the reader is advised to seek the manuscript by Georgescu et al (2013). Topology optimization of multi-physics problems is a much less researched topic, especially compared to structural topology optimization. This may be because Computational Fluid Dynamics (CFD) is a much more computationally intensive task compared with FEA. Recent studies have shown the potential of LBM methods in multi-physics topology optimization (Pingen et al (2007, 2009); Makhija et al (2012); Laniewski-Wollk and Rokicki (2016); Munk et al (2017, 2018a)). Further, since the LBM operates on a finite difference grid, is explicit in nature and requires only next neighbor interaction, it is very suitable for implementation on GPUs (Tölke and Krafczyk (2008)). Tölke and Krafczyk (2008) demonstrate a very efficient implementation of a LBM in 3D on a GPU. They obtain an efficiency gain of up to two orders of magnitude with respect to the performance on a CPU. Kuznik et al (2010) implement a general purpose LBM code, with all steps of the algorithm running on the GPU, achieving up to one billion lattice updates per second using single-precision floating points. Further, they show that single-precision floating point arithmetic is sufficient having a 3.8 times speed-up compared to double-precision. GPU implementation of LBMs have been used for real-time visualization of Fluid-Structure Interactions (FSI) for two-dimensional problems (Garcia et al (2011)). The authors achieved a speed increase when using their GPU-LBM of 222 times compared with a one core and 78 times compared with a two core CPU. Schönherr et al (2011) compare two multi-thread based parallel implementations of the LBM on different hardware platforms: a multi-core CPU implementation and a GPU implementation. They show that the limiting factor for the speed of the Lattice Boltzmann simulation is the memory bandwidth. More recently, Obrecht et al (2013) present a multi-GPU LBM solver managing to run the solver on six GPUs in parallel. With this architecture, they observed up to 2.15×10^9 node updates per second for the 2D lid-driven cavity test case. Such a performance is comparable to large high performance clusters or massively parallel super computers, showing the potential of GPU implementation in LBMs. Delbosc et al (2014) showed that real-time compute capability and satisfactory physical accuracy are achievable by combing a lattice Boltzmann model with the parallel computing power of a GPU. Along these lines, Khan et al (2015) performed real-time simulations of indoor environments, demonstrating significant speed up when implementing a lattice Boltzmann method on a GPU compared with traditional CFD based large eddy simulations.

The aim of this article is to determine the feasibility of using a GPU-LBM code with a Bi-directional Evolutionary Structural Optimization (BESO) algorithm for real world multi-physics design problems. So far BESO algorithms have not been employed with GPU architectures (Aissa et al (2014)). Furthermore, GPU implementation in topology optimization is a very recent field of research and therefore more studies must be made to increase the application of topology optimization in real world design problems. To the best of the authors' knowledge this is the first time a multi-physics topology optimization problem with an LBM-FEA code is implemented in a GPU architecture and compared with a CPU version of the code. The speed-up and optimization results are compared and discussed giving new insights into the difficulties involved with GPU-implemented topology optimization.

2 Methodology

In this section the LBM for modeling the fluid dynamics is briefly introduced. This is followed by the mathematical definition of the topology optimization problem and the BESO method is then described. For further details on GPU computing, the LBM and BESO meth-

ods the reader should seek out the textbooks by Sanders and Kandrot (2010), Succi (2001) and Huang and Xie (2010), respectively.

2.1 Lattice Boltzmann modelling

The ability to simulate flows through the use of CFD has progressed considerably, reducing test requirements at a lower cost and risk. Furthermore, CFD has been used to simulate real-world phenomena (Li and Luo (2014)). In the case when engineering applications require resolving fluid interactions with high accuracy, or involve low Mach number flow, mesoscopic flows and complex geometrical arrangements, LBM offers an alternative CFD method rather than using the Navier-Stokes (NS) equations (Succi (2001)). Moreover, LBM has been applied to a wide range of applications from theoretical physics to real-world problems, and is expected to provide one of the next evolutions in the computational sciences (Aidun and Clausen (2010); Wang and Menon (2001)), notably for multi-scale simulation and optimization (Liu et al (2016); Li et al (2016)). The LBM is a memory-bound algorithm, which makes it suitable for the GPU architectures. GPU offers a computational environment with many processors. GPU-implemented LBM codes have been used on a variety of applications in the aerospace field (Wang and Menon (2001)), being competitive in both accuracy and execution speed. However, the LBM has not been used extensively in topology optimization algorithms (Munk et al (2017)). Furthermore, to the best of the authors' knowledge, a GPU-implemented LBM has not been coupled with and compared against CPU topology optimization codes. Therefore, this work couples a GPU-implemented LBM to a BESO algorithm and compares both the final design, in terms of objective, and the computational time to a CPU-implemented version of the code.

The LBM constructs kinetic models, based on Newton's laws, incorporating the essential physics of microscopic processes, such that one can correctly model the macroscopic processes. A finite number of molecules, whose motion is governed by Newton's laws of dynamics, are used to model the fluid. A discretized Boltzmann equation is solved by the LBM, which uses velocity distribution functions to represent macroscopic properties. Both collision, the interaction of two particles, and streaming, the movement of particles from one node to the nearest neighbor, are modeled by the discrete Boltzmann equation. The fundamental concept behind the LBM is to calculate the macroscopic quantities from the moments of the finite number of velocity distribution functions, which are obtained by solving the discrete Boltzmann equation. A D3Q19 lattice is used in this work, i.e. 3 dimensions and 18 moving particles per rest node. The total number of iterations used for the LBM simulations is 4000, since stability has been demonstrated and validated against NS simulations using a commercial code, ANSYS CFX (Djenidi and Moghtaderi (2006)), and experimental analysis (Moghtaderi et al (2006)). For a more in-depth overview of the LBM, interested readers should seek out the textbook by Succi (2001). For details on how the LBM and FEA are coupled, the reader is advised to seek out the previous works by the authors on this topic (Munk et al (2017, 2018a,b)).

2.2 Topology optimisation

The first optimization problem studied in this article is the compliance minimization, or stiffness maximization, of a micro fluidic mixer under fluid pressure loads with a structural volume constraint. Therefore, the objective is to find the distribution of a pre-defined amount

of material such that a design with maximum stiffness is obtained. Hence, the topology optimization problem can be mathematically stated as:

$$\begin{aligned} \text{Minimize:} \quad & \frac{1}{2}\mathbf{u}^T[\mathbf{K}]\mathbf{u} \\ \text{subject to:} \quad & [\mathbf{K}]\mathbf{u} = \mathbf{f} \\ & \sum_{i=1}^{n} x_i V_{e_i} \leq V \\ & \mathbf{x} = \{0,1\} \end{aligned} \tag{1}$$

where $\mathbf{x}$ is the vector of design variables, x_i, n is the total number of elements in the model and V is a predefined structural volume. Since the algorithm is discrete (Section 1) the design variables can only be equal to $x_i = 1$, representing solid material, or $x_i = 0$, representing fluid/void material.

The second optimization problem this article is concerned with is the vorticity maximization of micro fluidic mixers for a given Reynolds number and structural volume. Therefore, the objective is to find the topology of the mixer that gives the highest vorticity in the region of interest. Hence, the topology optimization problem for this case is mathematically formulated as follows:

$$\begin{aligned} \text{Minimize:} \quad & -\overrightarrow{\omega} \\ \text{subject to:} \quad & Re = Re_0 \\ & \sum_{i=1}^{n} x_i V_{e_i} \leq V \\ & \mathbf{x} = \{0,1\} \end{aligned} \tag{2}$$

where $\overrightarrow{\omega}$ is the vorticity of the flow in the region of interest, Re is the Reynolds number of the flow, and Re_0 represents a predefined Reynolds number. For this problem, a design variable of $x_i = 1$ represents fluid elements, whereas $x_i = 0$ represents solid elements.

2.2.1 Evolutionary structural optimisation

The original ESO algorithm is monotonic, i.e. elements can only be removed from the design domain (Xie and Steven (1993)). These early methods are based on the successive elimination of *inefficient* material, gradually evolving the design towards the optimum (Xie and Steven (1997)). Although the ESO method has been applied to a wide range of problems (Xie and Steven (1996); Steven et al (2000)), it is limited in two main ways. Firstly, as already mentioned, structure can only be removed from the design domain, consequently the initial model must be significantly over designed. Secondly, if structure is prematurely removed it cannot be recovered (Munk et al (2015)). Subsequent ESO methods, referred to as BESO, allow material to be re-admitted to the design domain (Querin et al (1998)). Modern BESO algorithms are convergent and mesh independent (Huang and Xie (2007)), simultaneously removing and adding material from and to the design domain until all constraints and a convergence criterion are satisfied. More recently, a further improvement to BESO methods introduced the use of *soft* material to model the void elements in the FEA (Huang and Xie (2009)), known as *soft-kill* BESO with the former being *hard-kill* BESO. This article uses a soft-kill BESO method coupled to a GPU-implemented LBM. This work builds on a recent study by the authors (Munk et al (2017)), which implemented a CPU version of the code, aiming to drastically improve computational efficiency, bring high-fidelity methods forward to the preliminary design stage.

2.2.2 Sensitivity analysis

In this study, two different objectives are considered (Section 2.2). The first is minimum compliance or maximum stiffness, used for structural optimization. In FEA the removal of an element results in a reduction in the stiffness of the structure which is equal to the element strain energy (Chu et al (1996)). This change is defined as the element sensitivity for the compliance minimization problem:

$$\alpha_e^{cmp} = \frac{\partial c}{\partial x_i} = \frac{1}{2} p x_i^{p-1} \mathbf{u}_e^T [\mathbf{K}]_e^0 \mathbf{u}_e \tag{3}$$

where c is the compliance, $p = 3$ is the penalization factor, the subscript e represents elemental values and superscript cmp and 0 represents a compliance objective and solid values, respectively. The element sensitivity (eq. 3) takes advantage of the SIMP material model (Bendsøe and Sigmund (1999)), where the Young's modulus, E, is modeled using a power law penalization method, as follows:

$$E(x_i) = E^0 x_i^p \tag{4}$$

In design-dependent load problems, as is the case here, changes in the structure lead to variations in the load vector. Therefore, this variation in the load vector must be considered in the sensitivity analysis (Munk et al (2017)). Thus, from the definition of the optimization problem (eq. 1) the sensitivity analysis for compliance minimization (eq. 3) can be updated such that the variation in the load vector is considered, as follows (Yang et al (2005); Munk et al (2017, 2018b))

$$\alpha_e^{cmp} = \frac{\partial c}{\partial x_i} = \frac{1}{2} p x_i^{p-1} \mathbf{u}_e^T [\mathbf{K}]_e^0 \mathbf{u}_e + p x_i^{p-1} \mathbf{u}_e^T \Delta \mathbf{f}_e \tag{5}$$

where $\Delta \mathbf{f}_e$ is the change in the element load vector between optimization iterations. Taking the isoparametric bilinear elements used in this work, the change in the load vector of one element for a fluid pressure load is found by (Munk et al (2017))

$$\Delta \mathbf{f}_e = \frac{1}{4} P_i A_i \{1,0,0,0,0,0,1,0,0,0,0,0,\ldots,0\}_{24\times 1}^T \tag{6}$$

where P_i and A_i are the pressure load and elemental area, respectively. Eq. 6 is applicable for cases where the flow travels in the x-direction and the structure is aligned perpendicular to the flow, as is the case in this study. If this is not the case then the vector defined in eq. 6 must be updated to match the loading conditions. This study takes advantage of a two-domain approach, i.e. the structural and fluid dynamics are solved separately, during every optimization iteration. This method is more flexible than a monolithic approach, which solves an adjoint problem to update the structure and fluid solutions together. Furthermore, a monolithic approach is problem specific unlike a two-domain approach, which can be applied to all FSI problems. In a previous study by the authors (Munk et al (2018b)), this approach was demonstrated to work well as both the structural and fluid dynamics were shown to converge, even allowing a relaxation of the coupling conditions.

The second objective considered in the paper is vorticity maximization (Section 2.2). Thus, the goal of this problem is, for a given Reynolds number, to increase the mixing of two fluid species. This is imperative for the operation of micro fluidic mixers, as their purpose is to efficiently mix two, or more, fluid species. Hence, since the flows have low Reynolds numbers, normally lower than 1000, vorticity is an accurate measure of the degree of mixing

as shown in the works of Woodfield et al (2003) and Moghtaderi et al (2006). The authors of this work recently developed a soft-kill BESO method for the vorticity maximization of fluids using the LBM (Munk et al (2017)). The circulation method for vorticity (Abrahamson and Lonnes (1995)) and the shape derivative given in Kasumba and Kunisch (2012) are used to derive the sensitivity number to solve this optimization problem. Therefore, the sensitivity number for the vorticity maximization is determined by:

$$\alpha_e^{vrt} = \max(\overrightarrow{\omega}) - \Delta\gamma_e^T x_i^{p-1} \Delta\gamma_e \tag{7}$$

where $\overrightarrow{\omega}$ is the vorticity of the flow, the superscript *vrt* represents the vorticity objective and $\Delta\gamma_e$ is the change of, Δ, the element velocity vector defined as:

$$\gamma_e = \left\{\Delta\gamma_x, \Delta\gamma_y, \Delta\gamma_z, \Delta W_x, \Delta W_y, \Delta W_z\right\}^T \tag{8}$$

where γ_x, γ_y, γ_z are the spatial components and W_x, W_y and W_z are the circulation components. For more information on the derivation of the sensitivity numbers (eqs. 5 and 8) and for a validation against meta-heuristic algorithms the reader is advised to seek out the previous study by the authors (Munk et al (2017)), which outlines the method in more detail.

2.2.3 Mesh dependency and convergence

In order to guarantee that a solution to the topology optimization problem (eq. 1 and 2) exists, some restrictions on the design must be introduced (Sigmund and Petersson (1998)). The sensitivity numbers can become discontinuous across the element boundaries, resulting in mesh dependency or checkerboarding (the repetition of solid and void material). A filter scheme is used, to smooth the element sensitivity numbers across the entire domain, alleviating the problem of mesh dependency and checkerboarding. The filter scheme is similar to that presented by Sigmund and Petersson (1998); however, nodal sensitivity numbers are used when calculating the updated element sensitivity numbers based on the surrounding structure. The nodal sensitivity numbers are found by taking the average of all the element sensitivity numbers that are connected to the node, thus:

$$\alpha_{n_j} = \sum_{i=1}^{M} w_i \alpha_{e_i} \tag{9}$$

where M is the number of elements connected to the j^{th} node and α_{e_i} is the i^{th} element sensitivity number (eq. 5 and 7). The weighting factor of the i^{th} element, w_i, is a function of the distance between the center of the i^{th} element and the j^{th} node, r_{ij}, thus:

$$w_i = \frac{1}{M-1}\left(1 - \frac{r_{ij}}{\sum_{i=1}^{M} r_{ij}}\right) \tag{10}$$

The nodal sensitivity numbers (eq. 9) are then used in the mesh independency filter to find the smooth element sensitivities. A filter radius, r_{min}, is defined to identify the nodes that will have an effect on the element sensitivity. The value of r_{min} must be large enough such that the associated sub-domain, Ω, covers at least one element. Furthermore, this value must remain constant for all mesh sizes. Nodes that are located inside Ω contribute to the smoothing of the element sensitivity, by:

$$\alpha_{e_i} = \frac{\sum_{j=1}^{N} w(r_{ij})\alpha_{n_j}}{\sum_{j=1}^{N} w(r_{ij})} \tag{11}$$

where N is the total number of nodes in the sub-domain, Ω, and $w(r_{ij})$ is the linear weighting factor, defined as:

$$w(r_{ij}) = r_{min} - r_{ij} \quad j = 1,2,\ldots,N \tag{12}$$

The filter scheme effectively addresses the mesh-dependency and checkerboard problems. However, the objective function and corresponding topology may not be convergent. In order to overcome this problem, Huang and Xie (2007) showed that when the sensitivity numbers (eq. 11) are averaged with their previous values the solution becomes steadier, thus:

$$\alpha_{e_i} = \frac{\alpha_{e_i}^{itr} + \alpha_{e_i}^{itr-1}}{2} \tag{13}$$

where itr is the current iteration number. Therefore, the updated sensitivity number includes the history of the sensitivity information from previous iterations.

2.2.4 Convergence Criteria

For every iteration the BESO algorithm defines a target volume, found by:

$$V_{itr+1} = V_{itr}\,(1 \pm ER) \tag{14}$$

where ER, the evolutionary ratio, is a percentage of the current structural volume, and increases or decreases V_{itr+1} towards the desired volume constraint, V, defined in eq. 1. This, in turn, sets the threshold, α_{th}, of the sensitivity numbers. Thus, solid elements are removed from the design domain when:

$$\alpha_{e_i} \leq \alpha_{th} \tag{15}$$

and elements are added back to the design domain when:

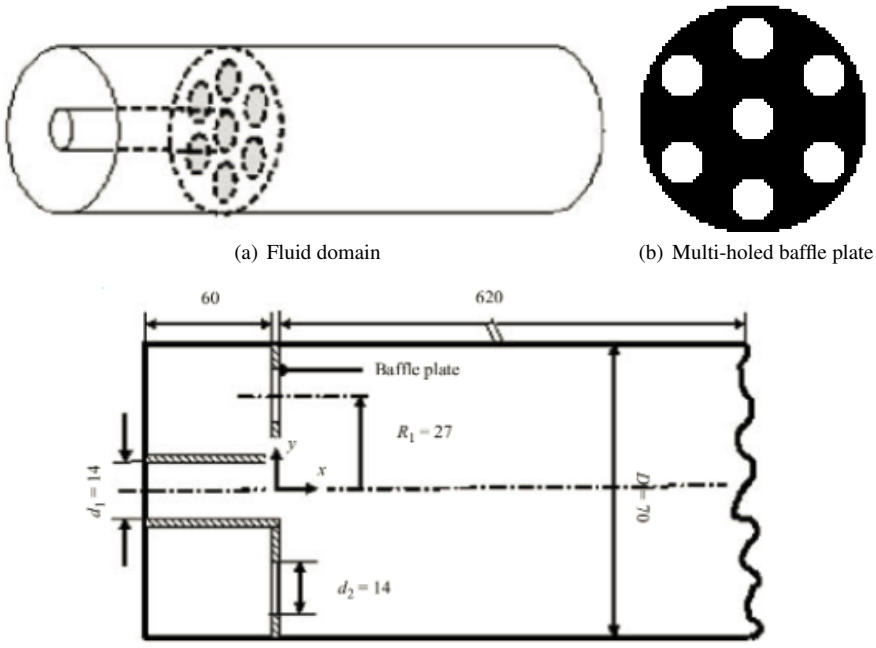
$$\alpha_{e_i} > \alpha_{th} \tag{16}$$

A maximum addition ratio, AR_{max}, is set to restrict the amount by which the volume of the structure can increase between iterations. Once $AR > AR_{max}$, the elements with the highest sensitivity numbers only are added, such that $AR = AR_{max}$. Then, in order to satisfy the target volume V_{itr+1}, the elements with the lowest sensitivity numbers are removed.

The iteration target volume remains constant at V once the volume constraint is satisfied. The topology evolves until a convergence criterion is satisfied. This is defined as:

$$\Delta O = \frac{\sum_{k=0}^{4} O_{itr-k} - \sum_{k=5}^{9} O_{itr-k}}{\sum_{k=0}^{4} O_{itr-4}} \leq \delta \tag{17}$$

where δ is a predefined tolerance, O is the objective function and itr is the current iteration of the optimization algorithm. Eq. 17 evaluates the change in the objective for the last 10 solutions. Therefore, if the change in the objective is minimal the solution is said to be converged. For a more in-depth discussion on evolutionary structural optimization algorithms, one should consult the latest textbook (Huang and Xie (2010)) and review paper (Munk et al (2015)) on the subject.



(a) Fluid domain (b) Multi-holed baffle plate

(c) Lay-out of micro-reactor model

Fig. 1 Baffled micro-reactor used in this study (Tsotskas et al (2015))

3 Case study

A baffled micro-reactor is used in this study as depicted in Fig. 1. The model is made up of a main pipe which is fitted with a fuel inlet tube, running along the axis of the main pipe, and a multi-holed baffle, which is where the secondary flow is introduced to the main flow. The lay-out and initial topology of the baffle are shown in Fig. 1.

The fluid domain (Fig. 1(c)) is defined in LBM nodes, here the lattice used has dimensions $680 \times 73 \times 73$ lattice units, with additional nodes used for the wall, in the x, y and z directions, respectively. The location of the baffle in the main pipe is 60 lattice units downstream of the flow inlet (Fig. 1(c)). The inlet boundary condition imposed is the velocity of the flow in the inlet tube and annulus area. The outlet boundary condition has a convective boundary condition, based on the velocity, applied. A no-slip condition is implemented along the walls by modeling them as full-way bounce-back. To mimic the experiments performed by Moghtaderi et al (2006), the difference in the mass flow rate between the inner tube and annulus area is set to 5%.

In the FEA a clamped boundary condition is applied along the perimeter of the baffle. Non-designable material is designated for the central hole boundary, since this is determined by the fuel line and inlet conditions, which have been fixed in the flow domain (Fig. 1(a)) to be consistent with the previous studies (Djenidi and Moghtaderi (2006); Moghtaderi et al (2006)).

The CPU simulations are performed on an Intel(R) Core(TM) i7-2720QM CPU 2.20GHz using 4 cores in parallel. The GPU simulations are performed on a Tesla M2070 with 5375 MB of total global memory, 448 available cores, 1150 MHz of stream processor rate and

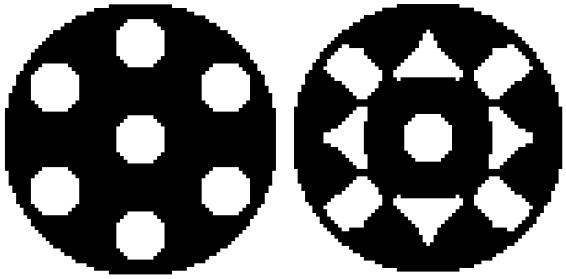
1566 MHz of memory clock rate. However, the speed of a simulation on the GPU is affected by the number of threads, which are created in the GPUs. Therefore, the specification of the hardware automatically calculates the number of threads by using the interfacing features of CUDA to query the provided GPU. Here, the solver is instructed to use 512 threads per block/kernel for a single simulation, which is one of the fastest settings provided by the current GPU.

4 Compliance minimization

In this section, topology optimization (Sect. 2.2) is applied to the multi-holed baffle plate (Fig. 1(b)) to maximize its stiffness for a given volume fraction. First, the CPU results are presented. These results are used as the benchmark for the GPU-implemented solutions. This is followed by the results of the single-precision GPU implantation of the code. In the literature of GPU-implemented FEA codes, it has been shown that using single-precision can causes numerical issues which result in convergence issues (Zegard and Paulino (2013)). Therefore, a double-precision version of the code is implemented and compared with the CPU and single-precision GPU results as well. However, it must be noted that while double-precision improves the numerical accuracy compared to single-precision, it only partially corrects these numerical issues as has been shown in Taufer et al (2010) and Demmel and Nguyen (2015).

4.1 CPU implementation

The CPU-LBM code is coupled with the BESO algorithm (Sect. 2.2). The optimization parameters: evolutionary ratio, $ER = 0.02$, volume fraction, $V = 0.58V_0$, maximum addition ratio, $AR_{max} = 0.02$, and tolerance, $\delta = 0.001$; are defined before the BESO algorithm is applied. The CFD mesh has 21,742,320 degrees of freedom. The initial and final structure is shown in Figure 2.



(a) Initial structure (b) Final structure

Fig. 2 Initial and final topology found using the CPU-LBM BESO algorithm

The final design obtained using the CPU code has been validated in previous numerical studies (Munk et al (2017, 2018a)). The compliance of the initial structure is $5.13 \times 10^9 Nm$, whereas the final structure has a compliance of $2.281 \times 10^9 Nm$. Therefore, the

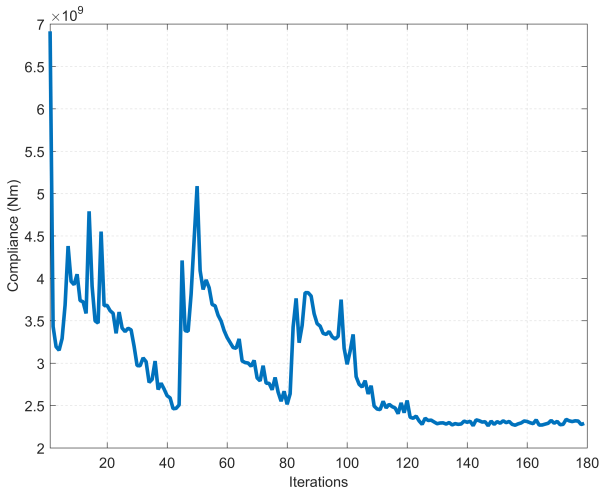


Fig. 3 Convergence history for the compliance minimization problem using the CPU-LBM BESO algorithm

CPU-implementation of the code is able to reduce the compliance by approximately 56%. The convergence history for the CPU algorithm is shown in Figure 3.

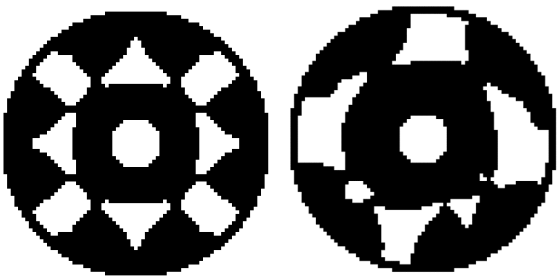
The algorithm takes 179 iterations to converge to the final solution (Fig. 3) when performed on a CPU. The computation time is approximately 7 days and 11 hours to complete the optimization. This is mainly due to the computational burden of the LBM, which has to be run 179 times. Typically, at the preliminary design stage, hundreds of design variations are being considered. Thus, at this computational expense it would take the CPU-implementation years to run all the cases. Hence, this is not a viable option and could only be used for the last iterations later in the design process.

4.2 Single-precision GPU implementation

The most computational efficient version of the multi-physics topology optimization algorithm studied in this work makes use of the single-precision GPU-LBM. However, one finds in the literature, on GPU-implemented topology optimization algorithms, cases where single-precision results in a lack of convergence, due to the inherent round-off errors (Martinez-Frutos and Herrero-Perez (2017)). Thus far, GPU-implemented topology optimization has been confined to structural optimization, hence GPU-FEA codes are producing these errors (Sec. 1). Therefore, it is expected in this study to observe similar errors when a single-precision code is used, since this study extends the GPU implementation to multi-physics problems. Namely, GPU fluids and CPU structures.

The single-precision GPU-LBM code is coupled with the BESO algorithm (Sect. 2.2). The optimization parameters are identical to the previous case (Sect. 4.1). The final structure determined by CPU and single-precision GPU is shown in Figure 4.

The final structure obtained using the single-precision GPU code is clearly not optimal (Fig. 4). Firstly, the initial structure (Fig. 2(a)) has a symmetry about the x- and y-axis, which is lost with the use of the single-precision GPU code. Further, it is known that the optimal structure for this particular problem should be symmetric, since the physics of the problem



(a) CPU (b) Single-precision GPU

Fig. 4 Final topology found using the single-precision GPU-LBM BESO algorithm and comparison with the CPU optimum

does not contain any unsymmetrical behavior (Munk et al (2017, 2018a)). Secondly, there is clear evidence of numerical errors in the topology (Fig. 4(b)), shown by the small holes that have formed, which are not present in the final topology of the CPU code (Fig. 4(a)). The compliance of the final structure found using the single-precision GPU code is $3.912 \times 10^9 Nm$, which is 71.5% increase from the final compliance of the structure found by the CPU code. The convergence history of the single-precision GPU algorithm is shown in Fig. 5.

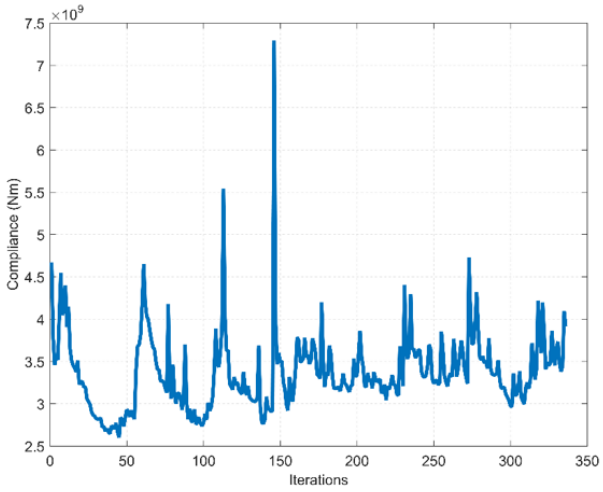


Fig. 5 Convergence history for the compliance minimization problem using the single-precision GPU-LBM BESO algorithm

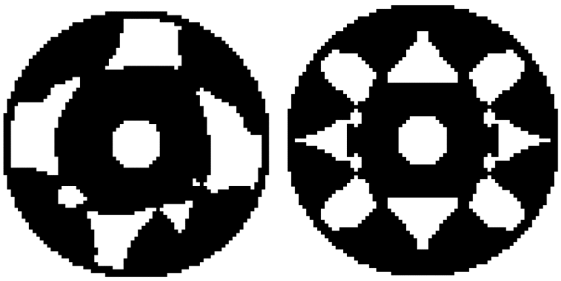
Clearly convergence is never achieved by the single-precision GPU algorithm, which stops after 336 iterations (Fig. 5). This equates to a computational time of approximately 13 hours, 31 minutes and 48 seconds. However, while this is a significant improvement over the computational expense of the CPU algorithm, over 13 times faster, the final result is not a feasible optimum. One could take the *best* solution found by the algorithm, in this

case at iteration 47 a solution was found having a compliance of around 2.6×10^9Nm, but convergence is never achieved and thus it is unlikely this solution is an optimum. Alternatively the *best* solution found could be used as an initial structure for an algorithm that is known to converge, possibly speeding up the overall process. Therefore, the single-precision GPU-driven topology optimization code, of this work, cannot guarantee convergence for the multi-physics compliance minimization problem. This confirms the conclusions of previous studies, which have observed the same phenomena for GPU-implemented SIMP algorithms on structural optimization problems (Zegard and Paulino (2013); Martinez-Frutos and Herrero-Perez (2017)). Thus, double-precision must be used.

4.3 Double-precision GPU implementation

Double-precision GPU codes are not as computationally efficient as single-precision GPU codes; however, they have a lower round-off error due to a higher floating point capacity. This is also the cause of the reduction in computational efficiency, since more memory is needed to store a higher amount of floating points. The GPU used in this work has a 2.0 compute capability, meaning that the floating point computation abides by the IEEE 754 standard for floating point arithmetic. Hence, round-off errors should be kept to a minimum.

The double-precision GPU-LBM code is coupled with the BESO algorithm (Sect. 2.2). The optimization parameters are again identical to the benchmark case (Sect. 4.1). The final structure determined by the single-precision and double-precision GPU codes is shown in Figure 6.



(a) Single-precision GPU (b) Double-precision GPU

Fig. 6 Final topology found using the double-precision GPU-LBM BESO algorithm and comparison with the single-precision GPU optimum

The final structure obtained using the double-precision GPU code shows a significant improvement over the single-precision GPU (Fig. 6). The structure is symmetric about both the x- and y-axis (Fig. 6(b)). Furthermore, no numerical errors are present in the structure, unlike the single-precision GPU code (Fig. 6(a)). Moreover, clear similarities between the final structure found using the CPU code (Fig. 2(b)) and the structure found using the double-precision GPU code are observed. The compliance of the final structure found using the double-precision GPU algorithm is $2.577 \times 10^9 Nm$, which is a 34% reduction compared with the single-precision GPU code and only a 12% increase compared with the CPU code. This increase in compliance is significant; however, by observing the convergence history of the CPU algorithm (Fig. 3) it is clear that the double-precision GPU algorithm is converging

to a local optimum. This is evident by comparing the final compliance found by the double-precision GPU algorithm with the multiple convergence cycles of the CPU algorithm (Fig. 3). Moreover, in a recent study by the authors (Munk et al (2018b)), it was shown that several local optima for this problem exist. Therefore, it is well known that one cannot criticize a gradient based optimization method for finding a locally optimal solution. The convergence history of the double-precision GPU algorithm is shown in Figure 7.

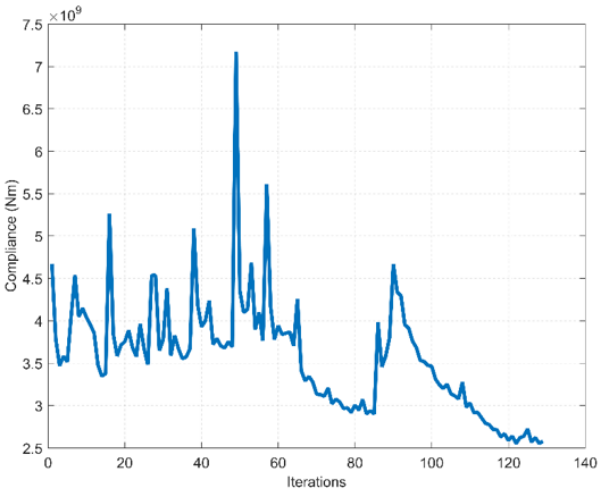


Fig. 7 Convergence history for the compliance minimization problem using the double-precision GPU-LBM BESO algorithm

Unlike the single-precision GPU code, convergence is achieved for the double-precision GPU code (Fig. 7). Furthermore, the double-precision topology optimization algorithm only takes 129 iterations to converge, which is less than the 179 iterations required by the CPU algorithm (Sec. 4.1). Hence, the computational time required to achieve convergence is approximately 10 hours, 1 minute and 48 seconds. Hence, 1 optimization iteration takes approximately 280 seconds, which is about double the time the single-precision topology optimization code takes (145 seconds). Therefore, the double-precision GPU code is implemented efficiently. Moreover, the double-precision GPU code is about 18 times faster than the CPU code, with a 12% reduction in objective. Hence, Pareto's principle of design is applicable here - 80% of the design comes from 20% of the time. Thus, this computational efficiency is more than beneficial at the preliminary design stages. Therefore, the double-precision GPU code could feasibly be used at the preliminary stages, whereas the CPU code could only be employed at the last stages in the design.

4.4 Difference between CPU and GPU implementation

The final analysis of this section is to quantify the difference between the CPU and GPU algorithms. Clearly, the CPU algorithm is able to find an optimal solution (Sec. 4.1), while the single-precision GPU is not (Sec. 4.2). The reason for this must lie in a discrepancy

between the two algorithms calculated sensitivity functions, which drive the design updates. Therefore, to quantify this discrepancy, the percentage difference of the sensitivity functions for the initial baffle topology using the CPU and GPU algorithms, i.e. $|\alpha_{CPU} - \alpha_{GPU}|/\alpha_{CPU}$, is illustrated in Fig. 8.

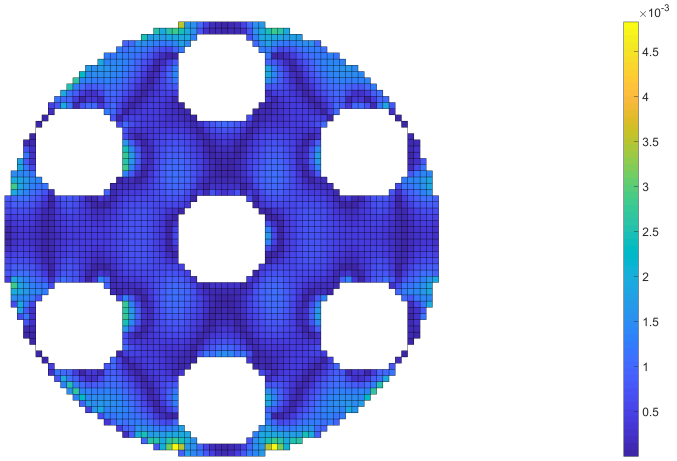


Fig. 8 Difference of the sensitivity functions determined by the CPU and GPU implementations

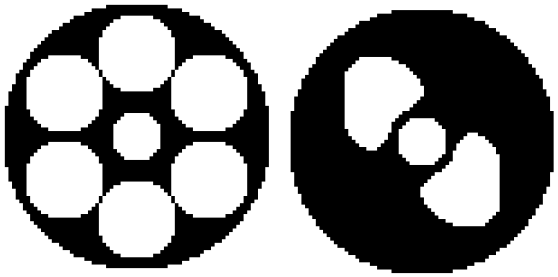
Clearly the discrepancies are kept to a minimum, having a maximum difference of 0.48% and an average difference of 0.038% (Fig. 8). However, one observation is that the distribution of the difference between the two sensitivity functions is not uniform and, more importantly, not symmetric. This explains why the GPU implemented algorithm becomes asymmetric, driving the solution to a non-optimal final design (Sec. 4.2). Therefore, even small variations, due to differences in computational architecture are able to perturb a system away from the optimum design. This is an important consideration when performing topology optimization on GPU-architectures.

5 Vorticity maximization

The second problem solved in this article applies the topology optimization algorithm (Sect. 2.2) to the multi-holed baffle plate (Fig. 2(b)) to maximize the amount of mixing between the two fluid species in the micro fluidic mixer. This problem is first solved using a CPU implementation of the code, to get a benchmark, which the GPU implementation can be compared against. A single-precision GPU-implemented code is then applied to the same problem. The previous section demonstrated how the lower floating point accuracy of single-precision GPU can lead to convergence errors in the topology optimization algorithm. Therefore, a double-precision GPU implementation is applied to the problem and compared against the CPU and single-precision GPU results.

5.1 CPU implementation

The CPU-LBM code is coupled with the BESO algorithm (Sect. 2.2). The optimization parameters are identical to that of the compliance minimization problem (Sect. 4). Furthermore, the CFD mesh is the same as for the compliance minimization problem, having 21,742,320 degrees of freedom. The initial and final structure is shown in Figure 9.



(a) Initial topology (b) Final topology

Fig. 9 Initial and final topology found using the CPU-LBM BESO algorithm for the vorticity maximization problem

The final design obtained using the CPU algorithm has been validated in previous numerical studies (Munk et al (2017, 2018a)). The vorticity of the initial topology is $4856s^{-1}$, whereas the final topology has a vorticity of $6060s^{-1}$. Therefore, the CPU implementation of the code increases the vorticity of the fluid by 25%. The convergence history for the CPU algorithm is given in Figure 10.

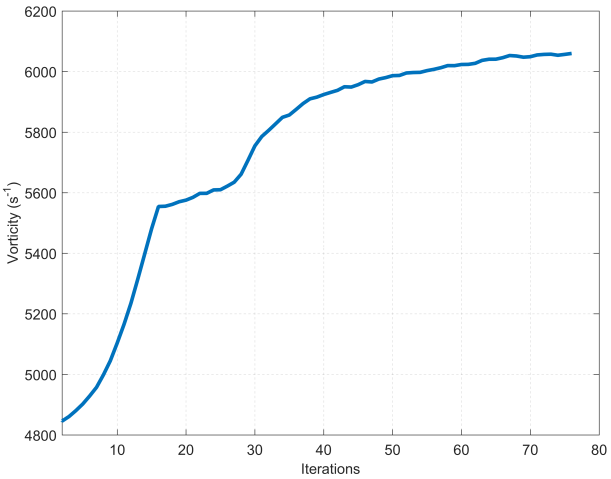


Fig. 10 Convergence history for the vorticity maximization problem using the CPU-LBM BESO algorithm

The algorithm takes 76 iterations to converge to the final solution (Fig. 10) when performed on a CPU. Therefore, the computation time is approximately 3 days and 4 hours to complete the optimization. Hence, similarly to the first topology optimization problem (Sect. 3), at the preliminary design stages this computation time is not viable, making the method only feasible for use later in the design cycle.

5.2 Single-precision GPU implementation

The single-precision GPU implementation has the fastest speed-up compared to the other methods, but has already been shown to produce numerical issues in the topology optimization algorithm (Sect. 4.2). This can only be attributed to the round-off errors inherent in single-precision GPU, since double-precision implementations have been able to achieve convergence (Sect. 4.3). Thus far, GPU-implemented BESO topology optimization algorithms have only been applied to structural objectives, namely compliance minimization. This section deals with a fluid objective, i.e. vorticity. Therefore, it is expected that the single-precision GPU implementation will not be able to solve this topology optimization problem, since the round-off errors are present in the formulation of the objective, i.e. the fluid properties, rather than the load application.

The single-precision GPU-LBM code is coupled with the BESO algorithm (Sect. 2.2). The optimization parameters are the same as is defined in the compliance minimization problem (Sect. 4). The final structure determined by CPU and single-precision GPU is shown in Figure 11.

(a) CPU (b) Single-precision GPU

Fig. 11 Final topology found using the single-precision GPU-LBM BESO algorithm for the vorticity maximization problem

As was expected, the final topology obtained by the single-precision GPU code is clearly not optimal (Fig. 11). Furthermore, the topology is not physically feasible, since there is structure which is suspended inside void material with no connection to the constraints. Moreover, in Munk et al (2017) it was demonstrated that the final topology had a symmetry about the $\pm 45^{\circ}$ diagonals. This symmetry is observed in the CPU result (Fig. 11(a)), whereas the single-precision GPU produces a final topology that is almost symmetric about the x- and y-axis. The final vorticity found using the single-precision GPU code is $5541 s^{-1}$, which is a 9% decrease compared to the final vorticity found by the CPU code. The convergence history of the single-precision GPU algorithm is shown in Fig. 12.

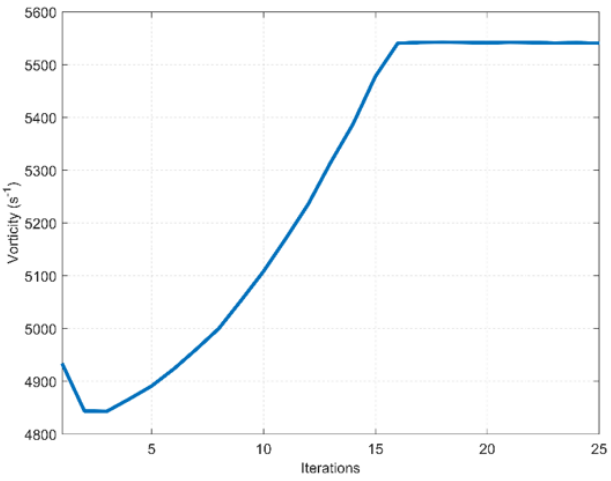


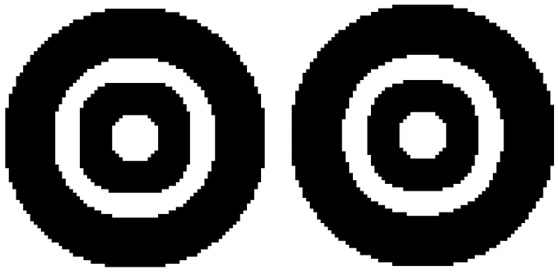
Fig. 12 Convergence history for the vorticity maximization problem using the single-precision GPU-LBM BESO algorithm

Unlike the compliance optimization problem (Sect. 4.2), convergence does seem to be achieved (Fig. 12). Furthermore, the algorithm only requires 25 iterations to achieve convergence, compared to 76 for the CPU implementation. This equates to a computational time of approximately 1 hour, which is 76 times faster than when a CPU algorithm is used. However, this notable improvement in computational expense is fruitless, since the final result is not a feasible optimum; although, it expressed convergence. Therefore, alike the compliance minimization problem, the single-precision GPU-implemented topology optimization code, of this work, cannot guarantee convergence to a feasible design for the vorticity maximization problem. Therefore, in the next section a double-precision GPU implementation is applied to the same problem to determine if the increase in numerical accuracy can produce a feasible optimum for a minimal increase in computational expense.

5.3 Double-precision GPU implementation

For the compliance minimization problem, it was found that the convergence issues occurring in the single-precision GPU implementation are avoided when double-precision is used. However, for the problem of this section, the single-precision GPU implementation does not seem to have convergence issues, but does not converge to a similar design found by the CPU implementation, or even a feasible one. Therefore, the double-precision GPU code may not be able to improve on the result found by the single-precision code. Nevertheless, the double-precision GPU implementation is applied to determine if round-off errors are the cause of the infeasible final design.

The double-precision GPU-LBM code is coupled with the BESO algorithm (Sect. 2.2). The optimization parameters are defined as in the compliance minimization problem (Sect. 4). The final structure determined by the single-precision and double-precision GPU codes is shown in Figure 13.



(a) Single-precision GPU (b) Double-precision GPU

Fig. 13 Final topology found using the double-precision GPU-LBM BESO algorithm for the vorticity maximization problem

The final topology obtained using the double-precision GPU code (Fig. 13(b)) is very similar to that found using the single-precision code (Fig. 13(a)). Again, the topology is almost symmetric about the x- and y-axis. Further, the final design is still not physically feasible since structure is suspended in void material. Unlike the compliance minimization problem, implementing double-precision in the GPU does not produce a feasible optimum. The vorticity of the final topology found using the double-precision code is $5553s^{-1}$, which is a 0.2% increase compared to the single-precision code and still a 8% decrease compared to the CPU code. The convergence history of the double-precision GPU algorithm is given in Figure 14.

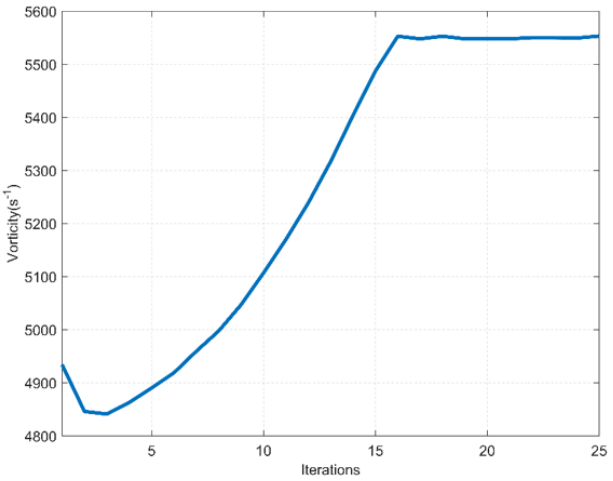


Fig. 14 Convergence history for the vorticity maximization problem using the double-precision GPU-LBM BESO algorithm

As would be expected, since the final designs are similar (Fig. 13), so too are the convergence histories of the single- and double-precision GPU-implemented topology optimization algorithms (Fig. 12 and 14). Again, the double-precision GPU code requires 25 iter-

ations to achieve convergence. Therefore, the computational time required to achieve convergence is approximately 1 hour, 56 minutes and 24 seconds, almost the double of what is required by the single-precision code. Hence, the double-precision code is about 39 times faster than the CPU implemented code. However, alike the single-precision GPU implementation the double-precision implementation fails to produce a feasible final design. Hence, unlike the compliance minimization problem (Sect. 4), this computational efficiency is not beneficial at the preliminary design stages, since infeasible designs are produced.

The results of this section indicate that there is an inherent difference between CPU and GPU architectures, which results in the optimizer converging to a different solution. This is troubling, since both CPU and GPU topology optimization codes started at the same initial design, had the same optimization parameters and both achieved convergence according to the same convergence criteria; however, they did not produce complementary results. This was not observed in the compliance minimization problem (Sect. 4), where the single-precision GPU code could not achieve convergence, but the double-precision code was able to converge to a design consistent with the CPU algorithm. One possibility is the difference in the reliance of the objectives, compliance and vorticity, on the GPU implementation. The objective, J, for compliance and vorticity is formulated as follows:

$$J(u) = \frac{1}{2}\mathbf{u}^T[\mathbf{K}]\mathbf{u} \tag{18}$$

$$J(\gamma) = \frac{1}{2}\int_{\Omega} |curl\gamma|^2 d\Omega \tag{19}$$

where Ω is the fluid domain, γ is the velocity field, $\mathbf{u}$ is the displacement field and $[\mathbf{K}]$ is the stiffness matrix of the structure. Therefore, since it is assumed that the structure stays within the elastic limit, i.e. does not undergo any plastic deformation, the following must hold:

$$\mathbf{F} = [\mathbf{K}]\mathbf{u} \tag{20}$$

where $\mathbf{F}$ is the force field applied to the structure. Hence, the compliance objective can be re-written as:

$$J(u) = \frac{1}{2}\mathbf{u}^T\mathbf{F} \tag{21}$$

Therefore, by comparing eq. 19 to eq. 21 it is noted that the reliance of the objectives on the GPU-LBM are different. For the compliance minimization problem the reliance is linear, since the applied force ($\mathbf{F}$) is determined by the LBM; whereas, for the vorticity objective it is more complicated, since the velocity field (γ) is determined by the LBM. The vorticity objective takes the square of the curl of this velocity field. Where the curl of a vector describes the infinitesimal rotation of the vector field. Therefore, it takes the difference of the partial differential in the three-spacial dimensions at every point in the vector field, mathematically this is described as:

$$curl\gamma = \left(\frac{\partial\gamma_z}{\partial y} - \frac{\partial\gamma_y}{\partial z}\right)\mathbf{i} + \left(\frac{\partial\gamma_x}{\partial z} - \frac{\partial\gamma_z}{\partial x}\right)\mathbf{j} + \left(\frac{\partial\gamma_y}{\partial x} - \frac{\partial\gamma_x}{\partial y}\right)\mathbf{k} \tag{22}$$

where $\mathbf{i}$, $\mathbf{j}$ and $\mathbf{k}$ are unit vectors for the x-, y- and z-axes. As pointed out in the CUDA programming guide (Nguyen H. (2007); NVIDIA Corporation (2008)) CUDA implements division and square root operations that are not IEEE-compliant, i.e. their error in units in last place is non-zero. However, addition and multiplication are IEEE-compliant. Therefore, any discrepancies between the CPU and GPU are greater in the vorticity objective than

in the compliance objective since division and square root operations are involved in the calculation of the objective and sensitivity functions.

6 Constrained topology optimization

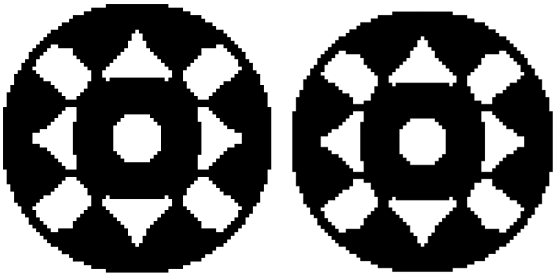
It is sometimes possible, by looking at the physics of the topology optimization problem, to determine certain *required* conditions, which can be directly enforced as constraints on the optimizer. This reduces the design space of the optimization problem and can often assist the algorithm in finding an optimum solution, or reduce its computational expense. Therefore, in this section two different constraints are applied, separately, to both the single-precision and double-precision GPU implementations to try and improve the designs obtained. First, it was shown that for the compliance minimization problem (Sect. 5.2) a symmetry about the x- and y-axis is an inherent feature of any feasible optimum to this problem (Munk et al (2017)). This is due to the lack of any unsymmetrical physical drivers in the system. Therefore, a symmetry constraint is implemented. Similarly, for the vorticity maximization problem, it was found that the CPU result has a symmetry about the $\pm 45^{\circ}$ diagonals (Sect. 5.1); however, is not symmetric about the x- and y-axis. Therefore, a symmetry constraint is applied to this problem, ensuring that this symmetry is enforced. Finally, it was demonstrated that for the vorticity maximization problem, physically infeasible final designs are produced by the single- and double-precision GPU implementations. Therefore, the last analysis of this section derives a novel feasibility constraint and applies it to the vorticity maximization problem only, since the GPU implementations of the compliance minimization problem do not produce infeasible designs.

6.1 Symmetry constraint

6.1.1 Compliance minimization

First, a symmetry constraint is applied to the compliance minimization problem for the single-precision GPU implementation. This constraint simply takes advantage of the symmetry of the problem, by taking only the top left quarter of the structure, at the end of every optimization loop, and then reflecting it about the z and then y-axis (Fig. 1(c)) to create the bottom half and right side of the structure for the next optimization loop. The final structure determined by CPU without a symmetry constraint and single-precision GPU with a symmetry constraint is shown in Figure 15.

The final structure obtained using the single-precision GPU code with a symmetry constraint is comparable to the CPU final design (Fig. 15). The symmetry constraint has forced the optimizer to only consider designs which are symmetric about the x- and y-axes. This restriction on the design domain is completely valid, since it is known before the optimizer is run that symmetry is a requirement of the final design (Munk et al (2017)). Therefore, by adding the symmetry constraint we are not restricting the optimizer, rather ensuring that it only considers physically feasible designs. The compliance of the final design found using the single-precision GPU code with a symmetry constraint is $2.416 \times 10^9 Nm$, which is only a 6% increase from the final compliance of the structure found using the CPU implementation ($2.281 \times 10^9 Nm$). Furthermore, the compliance is 6% less than the compliance of the final structure found using the double-precision GPU implementation. Hence, simply adding a symmetry constraint has produced a more optimum design than using a double-precision



(a) CPU no symmetry constraint (b) Single-precision GPU symmetry constraint

Fig. 15 Final topology found using the CPU and the single-precision GPU-LBM BESO algorithm without and with a symmetry constraint, respectively for the compliance minimization problem

GPU code instead of a single-precision. The convergence history of the single-precision GPU algorithm with a symmetry constraint is shown in Figure 16.

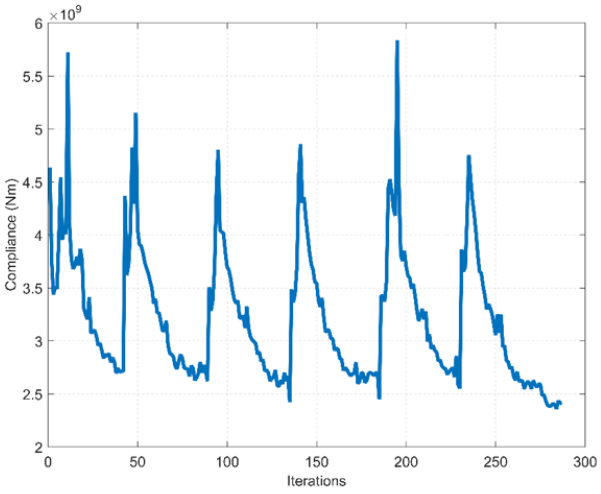


Fig. 16 Convergence history for the compliance minimization problem using the single-precision GPU-LBM BESO algorithm with a symmetry constraint

As is expected, since the final design is consistent with the CPU result, convergence is achieved for the single-precision GPU implementation with a symmetry constraint (Fig. 16). However, the single-precision GPU algorithm with a symmetry constraint takes 287 iterations to converge. Hence, the computational time required for convergence is approximately 11 hours, 33 minutes and 36 seconds, which is over an hour and half longer than the double-precision code without the symmetry constraint, due to the significant increase in required iterations. Nevertheless, this is still over 15 times faster than the CPU implementation, with only a 6% reduction in objective. The double-precision had a speed-up of 18 times, but with a 12% forfeit in objective. Therefore, the double-precision implementation is still the

most computationally efficient. A further observation is the large number of convergence cycles present in the convergence history (Fig. 16). This clearly indicates the presence of several local optima in the design space, the impact of this was discussed in the analysis of the results of the double-precision algorithm (Sec. 4.3). Therefore, it is no surprise that the different algorithms find different local optima. The symmetry constraint is not applied to the double-precision implementation for the compliance minimization problem since it produces a symmetric final design. Hence, symmetry is never broken and the constraint would never be violated. Therefore, applying the symmetry constraint will simply yield the same results as without the constraint.

6.1.2 Vorticity maximization

Next, the symmetry constraint is applied to the vorticity maximization problem for the single-precision GPU implementation. In this case, the initial structure of the problem is asymmetric about the x- and y-axis as this was found to be the case in the final design produced by the CPU implementation. Further, the symmetry constraint ensures that the structure is always symmetric about the $\pm 45^\circ$ diagonals. This is done by taking only the structure on one side of the 45° diagonal, at the end of each optimization loop, and reflecting it over this axis to create the structure on the other-side of the 45° diagonal for the next optimization loop. The final structure determined by CPU without a symmetry constraint and single-precision GPU with a symmetry constraint is shown in Figure 17.

(a) CPU no symmetry constraint (b) Single-precision GPU symmetry constraint

Fig. 17 Final topology found using the CPU and the single-precision GPU-LBM BESO algorithm without and with a symmetry constraint, respectively for the vorticity maximization problem

The final structure obtained using the single-precision GPU code with a symmetry constraint is comparable to the CPU design (Fig. 17). The symmetry constraint has forced the optimizer to only consider designs that are symmetric about both the $\pm 45^\circ$ diagonals and asymmetric about the x- and y-axis. Unlike the symmetry constraint on the compliance minimization problem, this restriction on the design is not physically valid as there is no physical reason to enforce it. Instead, we are using our knowledge of the CPU solution to assist the optimizer in finding a feasible design. The difference is, one would not have any reason to enforce this symmetry constraint without any prior knowledge of the solution. The final vorticity of the design found using the single-precision GPU code with a symmetry constraint is $6077 s^{-1}$, which is an increase of 0.3% from the vorticity of the final design

obtained using the CPU code ($6060s^{-1}$). This small increase in objective is due to the algorithm being guided to the right solution, therefore it is already in the neighborhood of the solution before the optimization begins. The convergence history of the single-precision GPU algorithm with a symmetry constraint is given in Figure 18.

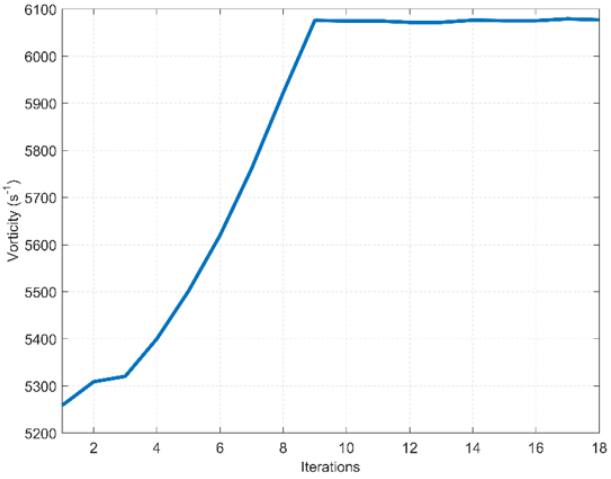


Fig. 18 Convergence history for the vorticity maximization problem using the single-precision GPU-LBM BESO algorithm with a symmetry constraint

The solution only takes 18 iterations to converge to the final design (Fig. 18). This further emphasizes that the solution is put on the right track by the symmetry constraint. This equates to a computational time of approximately 43 minutes and 30 seconds. Therefore, while this method would not be possible if no prior knowledge of the solution is known, it could be used after the CPU implementation has revealed certain features of the design, from earlier solutions, to speed up the solution of other design alternatives.

Finally, since the double-precision GPU implementation could not produce feasible design for the vorticity maximization problem, the symmetry constraint is applied. The final structure determined by the single-precision and double-precision GPU code with a symmetry constraint is shown in Figure 19.

As is expected the double-precision algorithm with a symmetry constraint produces an almost identical final topology to the single-precision algorithm with a symmetry constraint (Fig. 19). The final vorticity of the design found using the double-precision GPU algorithm with a symmetry constraint is $6084s^{-1}$, which is an increase of 0.1% from the vorticity of the final design using a single-precision algorithm with a symmetry constraint. The convergence history of the double-precision GPU algorithm with a symmetry constraint is shown in Figure 20.

The solution takes 19 iterations to converge to the final design (Fig. 20). This equates to a computational time of approximately 1 hour, 28 minutes and 48 seconds. This is just over double the time required by the single-precision GPU code with a symmetry constraint, with only a 0.1% improvement in objective. Therefore, in the case of a symmetry constraint

(a) Single-precision GPU symmetry constraint (b) Double-precision GPU symmetry constraint

Fig. 19 Final topology found using the single-precision and double-precision GPU-LBM BESO algorithm with a symmetry constraint for the vorticity maximization problem

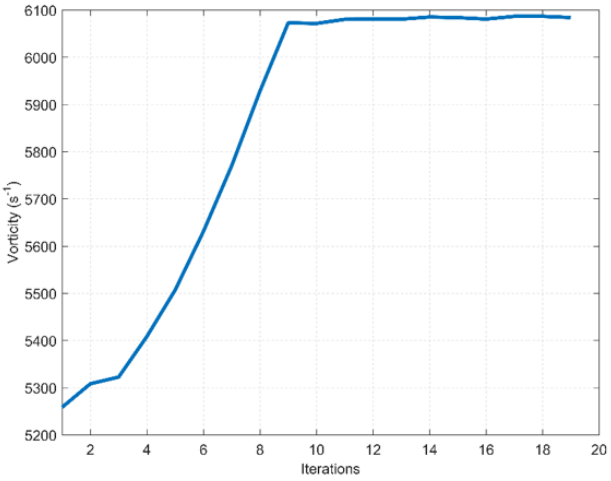


Fig. 20 Convergence history for the vorticity maximization problem using the double-precision GPU-LBM BESO algorithm with a symmetry constraint

the computational efficiency of the single-precision GPU code is superior to the increase in numerical accuracy of the double-precision GPU code.

6.2 Feasibility constraint

As mentioned earlier, the designs produced by the single- and double-precision GPU codes for the unconstrained vorticity maximization problem are not feasible. This is because the final topology contains structure suspended in void material. Therefore, in this section a feasibility constraint, which checks for structural islands is implemented to ensure feasible designs are produced. The feasibility constraint works by creating, what is termed here as, a connectivity matrix, $[\mathbf{A}]_{m\times n}$, where each entry, (m,n), in the matrix corresponds to an element on the baffle structure. Therefore, each entry contains a 1 if the corresponding element is solid and a 0 if the corresponding element is void. Hence, for each structure produced at

the end of each optimization loop an $[\mathbf{A}]$ is calculated. The closed structural boundaries, inside the baffle, for each structure are determined. Finally, the feasibility constraint sums the number of closed boundaries, n_{cb} inside the structure. If the number of closed boundaries is less than or equal to two, $n_{cb} \leq 2$, the volume constraint for that iteration is reduced and a new design is found. This process is repeated until a feasible design is produced for that iteration.

The feasibility constraint is applied to the single-precision GPU algorithm for the vorticity maximization problem with the same initial topology and optimization parameters as outlined in Section 5. The final design for the single-precision GPU algorithm with and without the feasibility constraint is shown in Figure 21.

(a) Without a feasibility constraint (b) With a feasibility constraint

Fig. 21 Final topology found using the single-precision GPU-LBM BESO algorithm without and with a feasibility constraint, respectively for the vorticity maximization problem

The final structure produced by the single-precision GPU is only feasible when the feasibility constraint is preformed (Fig. 21). There are no longer any structural islands present in the final design. Furthermore, unlike the symmetry constraint applied to the vorticity maximization problem, the restriction on the design domain is physically valid, since a feasible structure is a requirement of the final design. However, the final design is still noticeably different from the design found by the CPU algorithm (Fig. 13). This is because the design found using the single-precision GPU code with a feasibility constraint is still symmetric about the x- and y-axis, similar to that found without a feasibility constraint. The final vorticity of the design found by the single-precision GPU algorithm with a feasibility constraint is $5881s^{-1}$. This is a 3% reduction compared to the vorticity of the topology determined by the CPU algorithm ($6060s^{-1}$). The convergence history of the single-precision GPU algorithm with a feasibility constraint is given in Figure 22.

The solution takes 28 iterations to converge to the final design (Fig. 22). This equates to a computational time of approximately 1 hour, 7 minutes and 41 seconds, which is a speed-up of about 67 times compared with the CPU code. Therefore, for such a large increase in computational efficiency and only a 3% reduction in the objective, this is beneficial in the preliminary design stages where the CPU code is not viable due to its large computation time (over 3 days). Therefore, adding the feasibility constraint to the single-precision GPU algorithm has made the topology optimization method a viable option for use in the preliminary design stages, bringing these tools forwards in the design process.

Finally, the feasibility constraint is applied to the double-precision GPU algorithm to determine if the increase in the numerical accuracy can reduce the drop in the objective

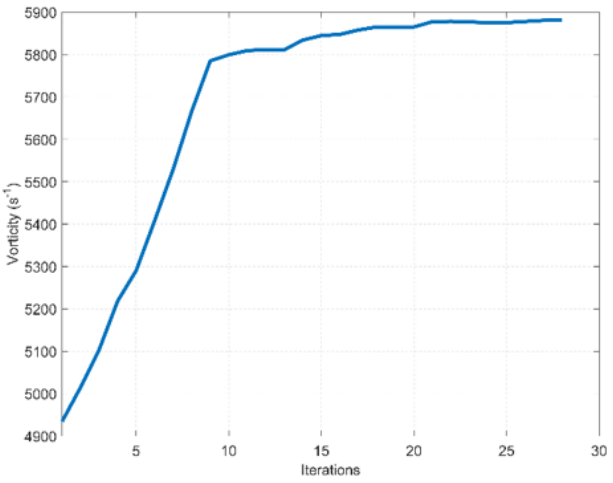


Fig. 22 Convergence history for the vorticity maximization problem using the single-precision GPU-LBM BESO algorithm with a feasibility constraint

from the CPU code. The vorticity maximization problem with the same initial topology and optimization parameters as outlined in Section 5 is solved. The final design for the single- and double-precision GPU algorithm with the feasibility constraint is shown in Figure 23.

(a) Single-precision (b) Double-precision

Fig. 23 Final topology found using the single- and double-precision GPU-LBM BESO algorithm with a feasibility constraint

The final design produced, with a feasibility constraint, by the double-precision GPU algorithm is almost identical to the design produced by the single-precision algorithm (Fig. 23). Hence, the design is still symmetric about the x- and y-axis when a double-precision GPU algorithm with a feasibility constraint is used, as was the case without the feasibility constraint. The vorticity of the final design found by the double-precision algorithm with a feasibility constraint is $5903s^{-1}$, which is only a 0.4% increase compared to the single-precision algorithm with a symmetry constraint ($5881s^{-1}$). The convergence history of the double-precision GPU algorithm with a feasibility constraint is given in Figure 24.

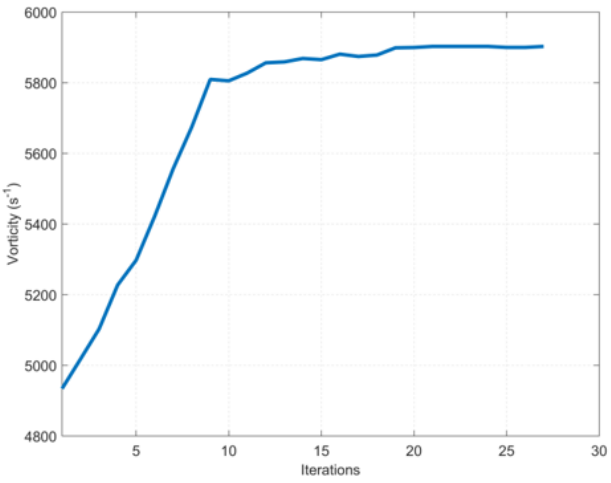


Fig. 24 Convergence history for the vorticity maximization problem using the double-precision GPU-LBM BESO algorithm with a feasibility constraint

The solution takes 27 iterations to converge to the final design (Fig. 24). This equates to a computational time of approximately 2 hours and 6 minutes, which is just under double the time required by the single-precision GPU code with a feasibility constraint. Thus in the case of the feasibility constraint, having only a 0.4% improvement in objective with twice the computation cost, the benefit of the computational efficiency inherent in the single-precision GPU code is superior to the increase in numerical accuracy of the double-precision GPU algorithm.

7 Summary

In this section a brief summary of the results of this work is given. The first problem solved was the compliance minimization problem (Sect. 4). It is demonstrated that the single-precision GPU implementation is unable to produce a feasible solution, having convergence issues. Therefore, a converged solution is never reached. The cause of this is the inherent round-off errors, which have also been observed in the literature for structural problems only (Martinez-Frutos and Herrero-Perez (2017)). However, it was shown that by implementing a double-precision GPU algorithm convergence could be achieved and a feasible optimum is found. Furthermore, the speed-up of the double-precision GPU algorithm is about 18 times compared with the CPU algorithm, having only a 12% reduction in objective. Therefore, it was concluded that the double-precision GPU implementation represent a feasible method that can be used at the preliminary design stages, whereas the CPU algorithm could not. Thus, for the compliance minimization problem, the implementation on a GPU has enabled these methods to be brought forward in the design cycle.

The next problem analyzed in this work was the vorticity maximization problem (Sect. 5). It is shown that neither the single- or double-precision GPU implementation could produce feasible solutions to this problem. However, unlike the compliance minimization problem, both implementations appear to converge. Thus why implementing double-precision

did not solve the problem. Moreover, it is demonstrated that the reliance of the topology optimization algorithm on the GPU numerics is greater in the vorticity maximization problem than in the compliance minimization problem, due to the different formulations of the objective functions.

In an effort to assist the GPU implementations additional constraints were formulated and added to the topology optimization problems. First, a symmetry constraint, which takes advantage of the symmetry of the physics of the compliance minimization problem, is implemented on the single-precision GPU algorithm. Since there are no physical asymmetric drivers, it is evident that the design of the baffle should be symmetric about the x- and y-axes (Munk et al (2017)). Therefore, by enforcing this symmetry, through the symmetry constraint, the single-precision GPU implementation is able to achieve convergence and produce optimum designs. Furthermore, the speed-up when compared with the CPU algorithm is about 15 times, with only a 6% reduction in objective.

Similarly, a symmetry about the $\pm 45^{\circ}$ diagonals and an asymmetry about the x- and y-axes was observed for the optimum solution to the vorticity maximization problem. Therefore, a symmetry constraint was employed, which enforced these conditions, in the single- and double-precision GPU implementations. This resulted in optimum final structures, similar to that found using the CPU algorithm, being produced. Furthermore, the computational efficiency is increased by 105 and 51 times compared with the CPU when using the single- and double-precision GPU algorithm, respectively. However, it was noted that this symmetry constraint was not valid, since the characteristics of the symmetry enforced were only known due to a prior knowledge of the solution. Thus, the symmetry constraint pushed the algorithm in the right direction. Nevertheless, this demonstrates how certain characteristics of the solution can be determined by running the slow CPU and then can be enforced in the fast GPU for quick optimization of other preliminary structures.

Finally, it was noted that the GPU implementations seemed to achieve convergence for the vorticity maximization problem; however, to infeasible designs. Therefore, a feasibility constraint was employed to ensure that the algorithms only considered feasible designs. By adding a feasibility constraint to the single- and double-precision GPU implementations for a vorticity objective, reasonable designs are produced without pushing the optimizer in the correct direction. Therefore, no pre-knowledge of the solution is required, making this a much more realizable solution compared to the symmetry constraint. Furthermore, the final design produces a similar final objective compared to the CPU result, having a reduction of 3% and 2% in objective for the single- and double-precision GPU implementation, respectively. However, this comes at a speed-up of 67 and 36 times for the single- and double-precision code, respectively. Thus, the small reduction in objective is worth the huge increase in computational efficiency. Again the double-precision code outperforms the single-precision. However, due to the increase in computational efficiency achieved by the single-precision code (about 46%) and the small improvement in objective by the double-precision code (about 0.4%) the single-precision code is more suited.

A quantitative comparison for all cases studied in this article is given in Table 1.

[1] convergence not achieved

Implementation	Iterations	Computational time	Objective
CPU (compliance)	179	7d 11h	$2.281(10^9)Nm$
CPU (vorticity)	76	3d 4h	$6060s^{-1}$
GPU SP (compliance)[1]	336	13h 31m	$3.912(10^9)Nm$
GPU SP (vorticity)	25	1h	$5541s^{-1}$
GPU DP (compliance)	129	10h 1m	$2.577(10^9)Nm$
GPU DP (vorticity)	25	1h 56m	$5553s^{-1}$
GPU SP sym (compliance)	287	11h 34m	$2.416(10^9)Nm$
GPU SP sym (vorticity)	18	43m 30s	$6077s^{-1}$
GPU DP sym (vorticity)	19	1h 29m	$6084s^{-1}$
GPU SP feasb (vorticity)	28	1h 8m	$5881s^{-1}$
GPU DP feasb (vorticity)	27	2h 6m	$5903s^{-1}$

Table 1 Numerical summary of results

8 Conclusion

A multi-physics topology optimization algorithm has been presented here for use in the preliminary design phases. The aim of this study is to use HPC methods to reduce the computational time required, such that these methods are viable for use at the preliminary design stages. A BESO algorithm is coupled to a GPU-enabled LBM flow solver to optimize the structural and flow characteristics of a micro-reactor. Hence, the process takes advantage of the high computational efficiency of GPU, which carried out the most computationally intensive part of the process, namely, the simulation of the flow via LBM. The implementation on both CPU and single- and double-precision GPU are performed and compared, determining the speed-up gained and loss in objective when different architectures are used. It was found that, for both topology optimization problems, implementation on a GPU resulted in a significant gain in computational efficiency, with only a small reduction in objective. Therefore, bringing these methods forward in the design cycle, where implementation on a CPU is not viable.

First, a multi-physics compliance minimization problem with design-dependent pressure loads was solved. It was found that the single-precision GPU implementation had convergence issues, and thus, was unable to find a suitable optimum. This phenomena has been noted in the literature for GPU-enabled FEA topology optimization (Martinez-Frutos and Herrero-Perez (2017)), but is a first here for multi-physics topology optimization. However, it is demonstrated that using a double-precision GPU implementation avoids these convergence issues, resulting in a speed-up of approximately 18 times, with only a 12% reduction in objective, compared to CPU.

Next, a vorticity maximization problem was solved on both single- and double-precision GPU. For this case, convergence is achieved; however, to infeasible designs, since structural islands are present for both single- and double-precision GPU. It was concluded that this discrepancy was due to the different reliance on the GPU numerics for the two different objectives.

Finally, a symmetry constraint and a feasibility constraint were added, separately, to the topology optimization problems to improve their convergence by eliminating infeasible designs from consideration. For the compliance minimization problem, adding a symmetry constraint, which takes advantage of the symmetrical physics, enabled the single-precision GPU implementation to converge to an optimum design. Similarly, for the vorticity maximization problem, adding a feasibility constraint forced only structurally feasible design to be considered by the optimizer, resulting in optimum designs being produced. Therefore, a

speed-up of about 67 times was achieved for the vorticity maximization problem, with only a 3% reduction in objective. Hence, the main findings of this article can be summarized by the following points:

- Single-precision GPU cannot be used without a symmetry constraint for the compliance minimization problem.
- Double-precision GPU produces better, in terms of objective, designs compared with single-precision for all cases.
- Single-precision is more computationally efficient for all cases, except compliance minimization problem.
- For all cases, computational time is drastically reduced with GPU implementation compared to CPU results.
- Adding a feasibility constraint to the vorticity maximization problem produces comparable designs for both the single- and double-precision GPU codes.

This study adds to the limited literature on GPU-accelerated topology optimization. New insights into the discrepancies between CPU and GPU numerics have been found, with reasonable methods developed to overcome these discrepancies. The work presented here brings high fidelity methods, such as Lattice Boltzmann flow simulations, coupled with rewarding optimization algorithms, such as topology optimization, forward to the preliminary design stages. This type of analysis is key to the continued application of topology optimization to real world aerospace design problems.

Acknowledgements

D. J. Munk thanks the Australian government for their financial support through the Endeavour Fellowship scheme.

The authors would like to acknowledge the UK Consortium on Mesoscale Engineering Sciences (UKCOMES) EPSRC grant No EP/L00030X/1 for providing the HPC capabilities used in this article.

References

Abrahamson S, Lonnes S (1995) Uncertainty in calculating vorticity from 2D velocity fields using circulation and least-squares approach. Exp Fluids 20:10–20

Aidun C, Clausen J (2010) Lattice-Boltzmann method for complex flows. Annual Review of Fluid Mechanics 42:439–472

Aissa M, Verstraete T, Vuik C (2014) Use of modern GPUs in design optimization. In: 10th ASMO-UK/ISSMO Conference on Engineering Design Optimization, Association for Structural and Multidisciplinary Optimization in the UK

Bendsøe M (1989) Optimal shape design as a material distribution problem. Struct Optimization 1(4):193–202, DOI 10.1007/BF01650949

Bendsøe M, Sigmund O (1999) Material interpolation schemes in topology optimization. Arch Appl Mech 69:635–654

Bendsøe M, Sigmund O (2003) Topology Optimization - Theory, Methods and Applications, 2nd edn. Berlin: Springer

Cecka C, Lew A, Darve E (2011) Assembly of finite element methods on graphics processors. International Journal for Numerical Methods in Engineering 85:640–669

Challis V, Roberts A, Grotowski J (2014) High resolution topology optimization using graphics processing units (GPUs). Struct Multidisc Optim 49(2):315–325

Chu D, Xie Y, Hira A, Steven G (1996) Evolutionary structural optimization for problems with stiffness constraints. Finite Elem Anal Des 21:239–251

D'Ammaro A, Kipouros T, Saddawi S, Savill A, Djenidi L (2010) Computational design for micro fluidic devices using Lattice Boltzmann and heuristic optimisation algorithms. In: In Joint OCCAM/ICFD Lattice Boltzmann Workshop, OCCAM/ICFD

Deaton J, Grandhi R (2014) A survey of structural and multidisciplinary continuum topology optimization. Struct Multidisc Optim 49:1–38, DOI 10.1007/s00158-013-0956-z

Delbosc N, Khan J, Kapur N, Noakes C (2014) Optimized implementation of the Lattice Boltzmann method on a graphics processing unit towards real-time fluid simulation. Computers and Mathematics with Applications 67:462–475

Demmel J, Nguyen H (2015) Parallel reproducible summation. IEEE Trans Comput 64(7):2060–2070

Djenidi L, Moghtaderi B (2006) Numerical investigations of laminar mixing in a coaxial microreactor. J Fluid Mech 568:223–243

Garcia M, Gutierrez J, Rueda N (2011) Fluid-structure coupling using Lattice-Boltzmann and fixed-grid FEM. Finite Elements in Analysis and Design 47:906–912

Georgescu S, Chow P, Okuda H (2013) GPU acceleration for FEM-based structural analysis. Archives of Computational Methods in Engineering 20(2):111–121

Herrero D, Martinez J, Marti P (2013) An implementation of level set based topology optimization using GPU. In: Proceedings of 10th World Congress on Structural and Multidisciplinary Optimization, WCSMO/ISSMO

Huang X, Xie Y (2007) Convergent and mesh-independent solutions for the bi-directional evolutionary structural optimization method. Finite Elem Anal Des 43:1039–1049

Huang X, Xie Y (2009) Bi-directional evolutionary topology optimization of continuum structures with one or multiple materials. Comput Mech 43:393–401

Huang X, Xie Y (2010) Topology Optimization of Continuum Structures: Methods and Applications, 1st edn. John Wiley & Sons

Kasumba H, Kunisch K (2012) Vortex control in channel flows using translation invariant cost functionals. Comput Optim Appl 52:691–717

Khan M, Delbosc N, Noakes C, Summers J (2015) Real-time flow simulation of indoor environments using Lattice Boltzmann method. Building Simulation 8:405–414

Kuznik F, Obrecht C, Rusaouen G, Roux JJ (2010) LBM based flow simulations using GPU computing processor. Computers and Mathematics with Applications 59:2380–2392

Laniewski-Wollk L, Rokicki J (2016) Adjoint Lattice Boltzmann for topology optimization on multi-GPU architecture. Computers and Mathematics with Applications 71:833–848

Li Q, Luo K (2014) Thermodynamic consistency of the pseudopotential lattice Boltzmann model for simulating liquid-vapor flows. Applied Thermal Engineering 72(1):56–61

Li Q, Luo K, Kang Q, He Y, Chen Q, Liu Q (2016) Lattice Boltzmann methods for multiphase flow and phase-change heat transfer. Progress in Energy and Combustion Science 52:62–105

Liu H, Kang Q, Leonardi C, Schmieschek S, Narváez A, Jones B, Williams J, Valocchi A, Harting J (2016) Multiphase lattice Boltzmann simulations for porous media applications. Computational Geosciences 20:777–805

Mahdavi A, Balaji R, Frecker M, Mockensturm E (2006) Topology optimization of 2D continua for minimum compliance using parallel computing. Struct Multidisc Optim 32(2):121–132

Makhija D, Pingen G, Yang R, Maute K (2012) Topology optimization of multi-component flows using a multi-relaxation time Lattice Boltzmann method. Comput Fluids 67:104–114

Martinez-Frutos J, Herrero-Perez D (2017) GPU acceleration for evolutionary topology optimization of continuum structures using isosurfaces. Computers and Structures 182:119–136

Martins J, Lambe A (2013) Multidisciplinary design optimization: A survey of architectures. AIAA Journal 59:2049–2075

Micikevicius P (2009) 3D finite difference computation on GPUs using CUDA. In: Proceedings of 2nd Workshop on General Purpose Processing on Graphics Processing Units, ACM

Moghtaderi B, Shames I, Djenidi L (2006) Microfluidic characteristics of a multi-holed baffle plate micro-reactor. Int J Heat Fluid Fl 27:1069–1077

Munk D, Vio G, Steven G (2015) Topology and shape optimization methods using evolutionary algorithms: A review. Struct Multidisc Optim 52(3):613–631, DOI 10.1007/s00158-015-1261-9

Munk D, Kipouros T, Vio G, Steven G, Parks G (2017) Topology optimization of micro fluidic mixers considering fluid-structure interactions with a coupled Lattice Boltzmann algorithm. Journal of Computational Physics 349:11–32

Munk D, Kipouros T, Vio G, Parks G, Steven G (2018a) Multiobjective and multi-physics topology optimization using an updated smart normal constraint bi-directional evolutionary structural optimization algorithm. Struct Multidisc Optim 57:665–688

Munk D, Kipouros T, Vio G, Parks G, Steven G (2018b) On the effect of fluid-structure interactions and choice of algorithm in multi-physics topology optimisation. Finite Elements in Analysis and Design 145:32–54

Nguyen H (2007) GPU Gems 3. Addison-Wesley Professional

NVIDIA Corporation (2008) NVIDIA CUDA - Programming Language. NVIDIA

Obrecht C, Kuznik F, Tourancheau B, Roux JJ (2013) Multi-GPU implementation of the Lattice Boltzmann method. Computers and Mathematics with Applications 65:252–261

Osher S, Sethian J (1988) Front propagating with curvature dependent speed: Algorithms based on Hamilton-Jacobi formations. Journal of Computational Physics 78(1):12–49

Pingen G, Evgrafov A, Maute K (2007) Topology optimization of flow domains using the Lattice Boltzmann method. Struct Multidisc Optim 36:507–524

Pingen G, Evgrafov A, Maute K (2009) Adjoint parameter sensitivity analysis for the hydrodynamic Lattice Boltzmann method with applications to design optimization. Comput Fluids 38:910–923

Querin O, Steven G, Xie Y (1998) Evolutionary structural optimization (ESO) using a bidirectional algorithm. Eng Comput 15:1034–1048

Rozvany G, Zhou M, Birker T (1992) Generalized shape optimization without homogenization. Struct Optimization 4(3):250–252, DOI 10.1007/BF01742754

Sanders J, Kandrot E (2010) CUDA by example: an Introduction to General-Purpose GPU Programming. Addison-Wesley Professional

Schmidt S, Schulz V (2011) A 2589 line topology optimization code written for the graphics card. Computing and Visulization in Science 14(6):249–256

Schönherr M, Kucher K, Geier M, Stiebler M, Freudiger S, Krafczyk M (2011) Multi-thread implementations of the Lattice-Boltzmann method on non-uniform grids for CPUs and GPUs. Computers and Mathematics with Applications 61:3730–3743

Sigmund O, Maute K (2013) Topology optimization approaches. Struct Multidisc Optim 48:1031–1055, DOI 10.1007/s00158-013-0978-6

Sigmund O, Petersson J (1998) Numerical instabilities in topology optimization: A survey on procedures dealing with checkerboards, mesh-dependencies and local minima. Struct Optim 16:68–75

Steven G, Li Q, Xie Y (2000) Evolutionary topology and shape design for general physical field problems. Comput Mech 26:129–139

Succi S (2001) The Lattice Boltzmann Equation for Fluid Dynamics and Beyond, 1st edn. Oxford University Press

Suresh K (2013) Efficient generation of large-scale Pareto-optimal topologies. Struct Multidisp Optim 47:49–61

Taufer M, Padron O, Saponaro P, Patel S (2010) Improving numerical reproducibility and stability in large-scale numerical simulations on GPUs. In: 24th IEEE International Symposium on Parallel and Distributed Processing (IPDPS), IEEE, pp 1–9

Tölke J, Krafczyk M (2008) TeraFLOP computing on a desktop PC with GPUs for 3D CFD. International Journal of Computational Fluid Dynamics 22:443–456

Tsotskas C, Kipouros T, Savill A (2014) The design and implementation of a GPU-enabled multi-objective Tabu-Search intended for real world and high-dimensional applications. Procedia Computer Science 29:2152–2161

Tsotskas C, Kipouros T, Savill A (2015) Fast multi-objective optimisation of a micro-fluidic device by using graphics accelerators. Procedia Computer Science 51:2237–2246

Vemaganti K, Lawrence WE (2005) Parallel methods for optimality criteria-based topology optimization. Comput Methods Appl Mech Engrg 194:3637–3667

Wadbro E, Berggren M (2009) Megapixel topology optimization on a graphics processing unit. SIAM Review 51(4):707–721

Wang H, Menon S (2001) Fuel-air mixing enhancement by synthetic microjets. AIAA Journal 39:2308–2319

Woodfield P, Kazuyoshi N, Suzuki K (2003) Numerical study for enhancement of laminar flow mixing using multiple confined jets in a micro-can combustor. Int J Heat Mass Transfer 46:2655–2663

Wu J, Dick C, Westermann R (2016) A system for high resolution topology optimization. IEEE Trans Vis Comput Graph 22:1195–1208

Xie Y, Steven G (1993) A simple evolutionary procedure for structural optimization. Comput Struct 49(5):885–896, DOI 10.1016/0045-7949(93)90035-C

Xie Y, Steven G (1996) Evolutionary structural optimization for dynamical problems. Comput Struct 58:1067–1073

Xie Y, Steven G (1997) Evolutionary Structural Optimization, 1st edn. Berlin: Springer

Yang X, Xie Y, Steven G (2005) Evolutionary methods for topology optimization of continuous structures with design dependent loads. Comput Struct 83:956–963

Zegard T, Paulino G (2013) Toward GPU accelerated topology optimization on unstructured meshes. Struct Multidisc Optim 48:473–485, DOI 10.1007/s00158-013-0920-y

https://dspace.org/

Multiphysics bidirectional evolutionary topology optimization on GPUarchitecture

Munk, David J.; Kipouros, Timoleon; Vio, Gareth A.

2018-10-16

https://doi.org/10.1007/s00366-018-0651-1

David J. Munk, Timoleon Kipouros and Gareth A. Vio. Multiphysics bidirectional evolutionary topology optimization on GPUarchitecture. Engineering with Computers, Volume 35, July 2019, pp. 1059-1079

Downloaded from a DSpace repository. https://dspace.org/