Massively parallel nodal discontinous Galerkin finite element method simulator for room acoustics

Melander, Anders; Strøm, Emil; Pind, Finnur; Engsig-Karup, Allan P.; Jeong, Cheol Ho; Warburton, Tim; Chalmers, Noel; Hesthaven, Jan S.

Published in:
International Journal of High Performance Computing Applications

Link to article, DOI:
10.1177/10943420231208948

Publication date:
2024

Document Version
Peer reviewed version

International Journal of
HIGH PERFORMANCE
COMPUTING APPLICATIONS

Research Paper

The International Journal of High Performance Computing Applications
2023, Vol. 0(0) 1–21

DOI: 10.1177/10943420231208948
journals.sagepub.com/home/hpc
Sage

Massively parallel nodal discontinous Galerkin finite element method simulator for room acoustics

Anders Melander[1], Emil Strøm[1], Finnur Pind[2,3], Allan P Engsig-Karup[1], Cheol-Ho Jeong[3], Tim Warburton[4], Noel Chalmers[4] and Jan S Hesthaven[5]

Abstract

We present a massively parallel and scalable nodal discontinuous Galerkin finite element method (DGFEM) solver for the time-domain linearized acoustic wave equations. The solver is implemented using the *libParanumal* finite element framework with extensions to handle curvilinear geometries and frequency dependent boundary conditions of relevance in practical room acoustics. The implementation is benchmarked on heterogeneous multi-device many-core computing architectures, and high performance and scalability are demonstrated for a problem that is considered expensive to solve in practical applications. In a benchmark study, scaling tests show that multi-GPU support gives the ability to simulate large rooms, over a broad frequency range, with realistic boundary conditions, both in terms of computing time and memory requirements. Furthermore, numerical simulations on two non-trivial geometries are presented, a star-shaped room with a dome and an auditorium. Overall, this shows the viability of using a multi-device accelerated DGFEM solver to enable realistic large-scale wave-based room acoustics simulations.

Keywords

High-performance computing, multi-device acceleration, heterogeneous CPU-GPU computing, room acoustic simulation, discontinuous Galerkin method

1. Introduction

Room acoustic simulations are widely used in, for example, building design, virtual reality, and hearing research. The task is challenging from a computational point of view as it involves simulating large and complex domains, over a broad frequency spectrum and long times. Room sizes range from 30 m^3 to 30,000 m^3, our auditory system can hear frequencies from 20 Hz to 20 kHz, and sound typically lasts in rooms somewhere between 0.5 s (living room) and 3.0 s (concert hall). Historically, the prevailing approach has been to apply *geometrical acoustics methods* (Savioja and Svensson, 2015), such as the image source method (Allen and Berkley, 1979) or ray-tracing (Krokstad et al., 1968), where the acoustic wave is approximated as a bundle of rays that are propagated in the room using the laws of ray optics. This reduces the computational task considerably, but the approximation is only appropriate in the high-frequency limit. At low-mid frequencies, wave phenomena such as diffraction and interference dominate and other simulation methods must be used. Recently, there also been development in utilizing scientific machine learning techniques such as physics-informed neural networks (PINNs) (Karniadakis et al., 2021) to generate inexpensive surrogate models to enable fast evaluations within room acoustics (Borrel-Jensen et al., 2021).

This motivates the use of *wave-based methods*, where the governing partial differential equations that describe wave motion are solved numerically. Given the right input data,

[1]Scientific Computing Section, Department of Applied Mathematics and Computer Science, Technical University of Denmark, Kgs. Lyngby, Denmark
[2]Henning Larsen, Copenhagen, Denmark
[3]Acoustic Technology Group, Department of Electrical Engineering, Technical University of Denmark, Kgs. Lyngby, Denmark
[4]Department of Mathematics, Virgina Tech, Blacksburg, VA, USA
[5]Chair of Computational Mathematics and Simulation Science, École Polytechnique Fédérale de Lausanne, Lausanne, Switzerland

Corresponding author:
Anders Melander, Department of Applied Mathematics and Computer Science, Technical University of Denmark, Anker Engelunds Vej 1, Kgs. Lyngby 2800, Denmark.
Email: anders.d.melander@gmail.com

these methods are very accurate, since no approximation to the wave propagation is made and all wave phenomena are inherently accounted for. Several different numerical techniques have been applied, such as the finite-difference time-domain (FDTD) method (Botteldooren, 1995), the pseudospectral time-domain (PSTD) method (Hornikx et al., 2016), the finite volume method (FVM) (Bilbao, 2013), the boundary element method (BEM) (Hargreaves et al., 2019), and the finite element method (FEM) (Craggs, 1994). Recently, nodal high-order accurate variants of the FEM, such as the spectral element method (SEM) (Pind et al., 2019) and the discontinuous Galerkin finite element method (DGFEM) (Wang et al., 2019), have been applied to simulate room acoustics. High-order methods result in better accuracy-per-computational-cost and they have the potential to significantly reduce the runtime of simulations and are a must for having good accuracy in wave propagation problems over long integration times (Kreiss and Oliger, 1972). Recent development has also looked at utilizing higher-order methods along with reduced order modeling to reduce computational cost (Sampedro Llopis et al., 2022). The nodal DGFEM is particularly well suited for room acoustic simulations, because it combines the attractive features of geometric flexibility, high-order accuracy, suitability for parallel computing, and lean memory usage. In the DGFEM schemes, the solution is computed locally for each element, with communication between elements only at the element boundaries. This data locality can be utilized for parallelization of the solver.

The major drawback of wave-based methods is the associated high computational cost. Simulating large spaces (larger than, say, 1000 m^3) into the mid frequency range (up to and beyond 1 kHz) typically requires tens or hundreds of millions of degrees of freedom (DoF). Despite progress in numerical methodology, wave-based methods are mostly applied to small rooms at very low frequencies. However, with the rapid advancements in many-core computing hardware, massively parallel high-performance computing (HPC)—in particular, the use of graphics processing units (GPUs) for scientific computing—offers a way to greatly extend the usability of wave-based methods to large spaces and for a broad spectrum of frequencies. This is due to the attractiveness of the on-chip performance of GPUs. The volume of calculations to data transfer ratio with the combination of higher-order methods such as DGFEM can exploit this high on-chip performance. While GPUs do bring their own challenges, such as more limited memory compared to usual CPU setups, this is compensated by the fact that strong performance can be achieved due to the FLOPS for free philosophy. These advantages of using GPUs mean that their use in wave-based room acoustics has been explored by several other authors. Lopez et al. (2013) and Spa et al. (2015) studied the GPU implementation of the FDTD method and found that, when implemented efficiently, the use of GPUs could drastically improve the computational performance. The FDTD method, like the DGFEM, lends itself naturally to parallelization but has a low arithmetic complexity. Takahashi and Hamada (2009) studied the GPU acceleration of the BEM when applied to the frequency domain Helmholtz equation and reported a considerable improvement in performance. Schoeder et al. (2019) presented a parallel CPU implementation of the acoustic wave function using DGFEM and also showed that DGFEM could achieve great computational scaling. Lastly, Morales et al. (2015) considered a distributed memory multi-CPU implementation of an adaptive rectangular decomposition (ARD) algorithm for room acoustic simulations and found that they could simulate large domains over broad frequency ranges using the HPC setup. While the ARD implementation of Morales et al. is described as efficient, it cannot handle complex geometries or non-trivial boundary conditions.

The goal of this work is to introduce the details of a HPC approach to simulate three-dimensional room acoustics using the DGFEM. To the best of our knowledge, this is the first time a massively parallel GPU-accelerated wave-based DGFEM room acoustic simulator is presented in the literature. The simulator is written in C/C++ with Message Passing Interface (MPI) as its backbone, and the device-interfacing is implemented using the single kernel language for parallel programming Open Concurrent Compute Abstraction (OCCA) (Medina et al., 2014) for portable multi-threading code development across different many-core hardware architectures. Curvilinear meshes and locally reacting frequency independent and frequency dependent impedance boundary conditions are supported in the implementation.

The paper is organized as follows. Section 2 presents the governing equations and the boundary conditions. Section 3 presents the numerical discretization of the governing equations. In Section 4, the HPC implementation is described and Section 5 contains several numerical experiments that offer insights into the performance of the simulator. Finally, some concluding remarks are offered in Section 6.

2. The mathematical model

Acoustic wave propagation in a lossless medium is governed by the coupled first order system of partial differential equations

$$\begin{aligned} \mathbf{v}_t &= -\frac{1}{\rho}\nabla p, \\ p_t &= -\rho c^2 \nabla\cdot\mathbf{v}, \end{aligned} \tag{1}$$

where $\mathbf{v}(\mathbf{x}, t)$ [m/s] is the particle velocity at position $\mathbf{x}$ in the domain Ω at time t, $p(\mathbf{x}, t)$ is the acoustic pressure [Pa], ρ is the density of air and c is the speed of sound in air. In this work, we use $\rho = 1.2$ kg/m^3 and $c = 343$ m/s. We let u, v, and w be the Cartesian coordinates x, y, and z parts of the velocity field $\mathbf{v} = [u,v,w]^T$.

2.1. Boundary conditions

To solve realistic problems, boundary conditions are needed, to accurately capture the absorption properties of the boundary of the domain $\partial\Omega$. Most real-world surfaces exhibit frequency dependent absorption, which can be enforced through locally reacting frequency dependent boundary conditions. Mathematically, this can be written as

$$\widehat{v}_n = \frac{\widehat{p}(\omega)}{Z(\omega)} = \widehat{p}(\omega)Y(\omega) \tag{2}$$

where $Y(\omega)$ is the normal boundary admittance, ω is the angular frequency, and $(\widehat{v}_n, \widehat{p})$ denote the Fourier transformed normal velocity and pressure, respectively. The absorption is dictated by the normal incidence surface impedance $Z(\omega)$, which is a complex function of the angular frequency. For a particular surface, $Z(\omega)$ can be measured, for example, in an impedance tube or using pressure-velocity sensors. Alternatively, several material models exist to estimate surface impedance of many common materials and constructions. Two special cases of these boundary conditions exists, the *rigid* case, where the right hand side is assumed to be zero,

$$\widehat{\mathbf{n}} \cdot \mathbf{v} = 0 \quad \text{on} \quad \partial\Omega \tag{3}$$

which defines a perfectly reflecting boundary, and the *frequency independent* case, where Z and Y are assumed constant

$$\widehat{\mathbf{n}} \cdot \mathbf{v} = v_n = \frac{p}{Z} = pY \tag{4}$$

To incorporate the frequency dependent boundary conditions accurately and efficiently into the time-domain simulation, the method of auxiliary differential equations (ADEs) can be used (Dragna et al., 2015; Pind et al., 2019). In the ADE method, the surface admittance is approximated as a rational function, defined as,

$$Y(\omega) = \frac{a_0 + \cdots + a_N(-j\omega)^N}{1 + \cdots + b_N(-j\omega)^N} \tag{5}$$

which can be rewritten using a partial fraction decomposition

$$\begin{aligned} Y(\omega) \quad &= Y_\infty + \sum_{k=1}^{Q} \frac{A_k}{\lambda_k - j\omega} + \\ &\sum_{k=1}^{S} \left(\frac{B_k + jC_k}{\alpha_k + j\beta_k - j\omega} + \frac{B_k - jC_k}{\alpha_k - j\beta_k - j\omega} \right). \end{aligned} \tag{6}$$

Here Q is the number of real poles, S is the number of complex conjugate pole pairs of $Y(\omega)$, λ_k are the real poles, $\alpha_k \pm j\beta_k$ are the complex conjugate pole pairs, and Y_∞, A_k, B_k, and C_k are numerical coefficients. For causality, λ_k and α_k must be semi-positive. Furthermore, for each set of numerical coefficients, the passivity of the multipole model must be checked. If the model is non-passive, it will add energy to the simulation, potentially leading to a blowup of the solution. The total number of poles $N_{\text{poles}} = Q + 2S$ should be chosen large enough to satisfy an error threshold of the multipole approximation of the boundary admittance.

By applying the inverse Fourier transform to (2), we get

$$v_n(t) = \int_{-\infty}^{t} p(t')y(t - t')dt' \tag{7}$$

Then by applying the inverse Fourier transform to (6) and inserting the result into (7), the expression for the velocity v_n at the boundary becomes

$$\begin{aligned} v_n(t) \quad &= Y_\infty p(t) + \sum_{k=1}^{Q} A_k \phi_k(t) + \\ &\sum_{k=1}^{S} 2\left(B_k \psi_k^{(1)}(t) + C_k \psi_k^{(2)}(t)\right), \end{aligned} \tag{8}$$

where ϕ_k is the accumulator for the k'th real poles, and $\psi_k^{(1)}$ and $\psi_k^{(2)}$ are the accumulators for the k'th complex conjugate poles pairs. These accumulators are defined by a set of ordinary differential equations (ODEs).

$$\begin{aligned} &\frac{d\phi_k}{dt} + \lambda_k \phi_k = p(t), \\ &\frac{d\psi_k^{(1)}}{dt} + \alpha_k \psi_k^{(1)} + \beta_k \psi_k^{(2)} = p(t), \\ &\frac{d\psi_k^{(2)}}{dt} + \alpha_k \psi_k^{(2)} - \beta_k \psi_k^{(1)} = 0. \end{aligned} \tag{9}$$

Thus, when using the ADE method, an additional set of ODEs must be solved at the boundary in each time step. The added computational cost due to this additional set of ODEs is relatively low, as the amount of extra ODEs needed to solve is small compared to the system that must be solved for the acoustic wave equations. In total, there will be N_{poles} times the number of boundary nodes with frequency

dependent boundary condition ODEs to solve. This extra work generally grows significantly slower than the total number of nodes in a domain when the mesh resolution increases in the domain.

3. The numerical model

3.1. Spatial discretization

The spatial derivatives in (1) are discretized using DGFEM. In this section, we will go through the discretization method. A more detailed description can be found in Hesthaven and Warburton (2008). DGFEM is a high-order method capable of solving the time-domain acoustic partial differential equations within a domain Ω, subject to an appropriate set of boundary and initial conditions. The spatial domain is tessellated by K non-overlapping elements as

$$\Omega \simeq \Omega_h = \bigcup_{k=1}^{K} D^k \tag{10}$$

In this work, the elements are taken to be tetrahedral elements in the three-dimensional space. On each of these elements the solution variables $\mathbf{v}$ and p are expressed on the k'th element in terms of polynomial basis functions and corresponding nodal coefficients. This is expressed as

$$\begin{aligned} \mathbf{v}_h^k(\mathbf{x},t) &= \sum_{n=1}^{N_p} \widehat{\mathbf{v}}_h^k(\mathbf{x}_n^k,t)\phi_n^k(\mathbf{x}) = \sum_{n=1}^{N_p} \mathbf{v}_h^k(\mathbf{x}_n^k,t)\ell_n^k(\mathbf{x}), \\ p_h^k(\mathbf{x},t) &= \sum_{n=1}^{N_p} \widehat{p}_h^k(\mathbf{x}_n^k,t)\phi_n^k(\mathbf{x}) = \sum_{n=1}^{N_p} p_h^k(\mathbf{x}_n^k,t)\ell_n^k(\mathbf{x}). \end{aligned} \tag{11}$$

Here $\phi_i^k(\mathbf{x})$ and $\ell_i^k(\mathbf{x})$ are the modal Jacobi and nodal Lagrange polynomial basis function on element D^k, respectively, and $\widehat{\mathbf{v}}_h^k(\mathbf{x}_n^k,t)$ and $\widehat{p}_h^k(\mathbf{x}_n^k,t)$ and $\mathbf{v}_h^k(\mathbf{x}_n^k,t)$ and $p_h^k(\mathbf{x}_n^k,t)$ are the modal and nodal coefficients, respectively. The modal basis functions are assumed to be orthonormal. On each element, $\mathbf{x}_n^k$ is a node distribution of $N_p = ((N+1)(N+2)(N+3))/6$ distinct nodal interpolation points, where N is the order of the basis functions. An α-optimized nodal distribution is used due to its low Lebesque constants (Warburton, 2006). The solutions on the elements are used to approximate the global solutions $\mathbf{v}$ and p in terms of direct sums

$$\mathbf{v} \approx \mathbf{v}_h = \oplus_{k=1}^{K} \mathbf{v}_h^k,\ p \approx p_h = \oplus_{k=1}^{K} p_h^k \tag{12}$$

The strong formulation of the governing equations, when using the nodal basis series to represent the unknown variables $\mathbf{v}_h^k$ and p_h^k, takes the following form

$$\begin{aligned} \int_{D^k} \frac{\partial \mathbf{v}_h^k}{\partial t}\ell_n^k(\mathbf{x})d\mathbf{x} &= -\frac{1}{\rho}\int_{D^k} \nabla p_h^k \ell_n^k(\mathbf{x})d\mathbf{x} \\ &\quad + \frac{1}{\rho}\int_{\partial D^k} \widehat{\mathbf{n}}\left(p_h^k - p^*\right)\ell_n^k(\mathbf{x})d\mathbf{x}, \\ \int_{D^k} \frac{\partial p_h^k}{\partial t}\ell_n^k(\mathbf{x})d\mathbf{x} &= -\rho c^2 \int_{D^k} \nabla\cdot\mathbf{v}_h^k \ell_n^k(\mathbf{x})d\mathbf{x} \\ &\quad + \rho c^2 \int_{\partial D^k} \widehat{\mathbf{n}}\cdot\left(\mathbf{v}_h^k - \mathbf{v}^*\right)\ell_n^k(\mathbf{x})d\mathbf{x}, \end{aligned} \tag{13}$$

where * denotes numerical fluxes that propagate the solution between elements and $\widehat{\mathbf{n}}$ is the outward pointing normal. This can be rewritten in a discrete formulation using matrix-based operators for integration and differentiation

$$\frac{du_h^k}{dt} = -\frac{1}{\rho}\mathscr{D}_x^k p_h^k + \frac{1}{\rho}\mathscr{L}^k\left(p_h^k - p^*\right)\widehat{n}_x, \tag{14a}$$

$$\frac{dv_h^k}{dt} = -\frac{1}{\rho}\mathscr{D}_y^k p_h^k + \frac{1}{\rho}\mathscr{L}^k\left(p_h^k - p^*\right)\widehat{n}_y, \tag{14b}$$

$$\frac{dw_h^k}{dt} = -\frac{1}{\rho}\mathscr{D}_z^k p_h^k + \frac{1}{\rho}\mathscr{L}^k\left(p_h^k - p^*\right)\widehat{n}_z, \tag{14c}$$

$$\begin{aligned} \frac{dp_h^k}{dt} &= -\rho c^2\left(\mathscr{D}_x^k u_h^k + \mathscr{D}_y^k v_h^k + \mathscr{D}_z^k w_h^k\right) \\ &\quad + \rho c^2 \mathscr{L}^k \widehat{\mathbf{n}}\cdot\left(\mathbf{v}_h^k - \mathbf{v}^*\right) \end{aligned} \tag{14d}$$

The matrix operations on the right hand side can be expressed in terms of the operators defined on a reference elements with coordinates $X = r, s, t$. This means we can avoid computing and storing each matrix for every element. We define the matrix operators on the reference element

$$\mathscr{D}_r = \mathscr{V}_r\mathscr{V}^{-1}, \mathscr{D}_s = \mathscr{V}_s\mathscr{V}^{-1}, \mathscr{D}_t = \mathscr{V}_t\mathscr{V}^{-1}. \tag{15}$$

Here $\mathscr{V}_{ij} \equiv \phi_j(\mathbf{r}_i)$ is the generalized Vandermonde matrix and $(\mathscr{V}_X)_{ij} \equiv \partial\phi_j(\mathbf{r}_i)/\partial X$ are defined on the reference element.

These operators can, via the chain rule of the continuous variables, for example,

$$\frac{\partial f(\mathbf{x})}{\partial x} = \frac{\partial r}{\partial x}\frac{\partial f(\mathbf{x})}{\partial r} + \frac{\partial s}{\partial x}\frac{\partial f(\mathbf{x})}{\partial s} + \frac{\partial t}{\partial x}\frac{\partial f(\mathbf{x})}{\partial t} \tag{16}$$

Be related to the corresponding discrete differential operations in the physical domain for each element as

$$\mathscr{D}_i^k = r_i^k\mathscr{D}_r + s_i^k\mathscr{D}_s + t_i^k\mathscr{D}_t, \text{for} \quad i = \{x,y,z\} \tag{17}$$

Furthermore, we introduce the lift operator $\mathscr{L}^k$ needed for the incorporation of boundary conditions via numerical fluxes at the element edges

$$\mathscr{L}^k = J^k(\mathscr{M})^{-1}\mathscr{E}, \mathscr{M} = \left(\mathscr{V}\mathscr{V}^T\right)^{-1}, \tag{18}$$

where $\mathscr{M}$ is the mass matrix defined on the reference element, J^k is the local transformation Jacobian, and $\mathscr{E}$ is the column-wise concatenation of mass matrices defined in terms of the nodal distributions on the faces.

The choice of the numerical flux stabilizes the dispersion and dissipation properties of the DGFEM scheme. In this work, the upwind flux is chosen. The upwind flux combines low dispersion errors over a wide range of frequencies with dissipation for underresolved frequencies (Ainsworth, 2004; Hu and Atkins, 2002). To derive the upwinding flux, a Riemann problem at each element edge must be solved. The derivation follows the work of Hesthaven and Warburton (2008) and can be seen in full in Appendix A. The upwinding flux takes the form

$$(\widehat{\mathbf{n}}F)^* = \begin{bmatrix} \frac{1}{2}\widehat{n}_x\left(c\left(u^m\widehat{n}_x + v^m\widehat{n}_y + w^m\widehat{n}_z\right) + \frac{1}{\rho}p^m\right) \\ \frac{1}{2}\widehat{n}_y\left(c\left(u^m\widehat{n}_x + v^m\widehat{n}_y + w^m\widehat{n}_z\right) + \frac{1}{\rho}p^m\right) \\ \frac{1}{2}\widehat{n}_z\left(c\left(u^m\widehat{n}_x + v^m\widehat{n}_y + w^m\widehat{n}_z\right) + \frac{1}{\rho}p^m\right) \\ \frac{\rho c^2}{2}\left(u^p\widehat{n}_x + v^p\widehat{n}_y + w^p\widehat{n}_z\right) + \frac{c}{2}p^p \end{bmatrix} \tag{19}$$

With $u^m = u^- - u^+$ and $u^p = u^- + u^+$, and u^- and u^+ denoting the value on the current and neighboring element, respectively. The same follows for the other variables.

Boundary conditions are weakly enforced through the numerical flux terms. For element, faces that lie on the domain boundary, $\mathbf{q}_h^+ = [u_h^+, v_h^+, w_h^+, p_h^+]$ needs to be defined according to the boundary conditions. For surface impedance boundary conditions, an averaging approach is used to impose the "Neumann-type" boundary conditions for the velocities, meaning the neighboring value is given as $p_h^+ = p_h^-$ and $\mathbf{v}_h^+ = -\mathbf{v}_h^- + 2\mathbf{g}_v$, where $\widehat{\mathbf{n}} \cdot \mathbf{g}_v = v_n$, with v_n given by either (4) or (8). For the computation of v_n, note that p corresponds to p_h^-, that is, the value on the current element. Thus, at the domain boundary nodes, the flux term in (14d) becomes $\widehat{\mathbf{n}} \cdot (\mathbf{v}_h^k - \mathbf{v}^*) = \widehat{\mathbf{n}} \cdot \mathbf{v}_h^- - v_n$.

3.2. Temporal discretization and stability

The semi-discrete system in (14a)–(14d) can be expressed as a general form ODE

$$\frac{d\mathbf{q}_h}{dt} = \mathscr{L}(\mathbf{q}_h(t), t) \tag{20}$$

where $\mathscr{L}$ is the spatial operator. This ODE system is integrated in time using a low-storage 4th order explicit Runge–Kutta (LSERK) time-stepping method,

$$\begin{aligned} &\mathbf{q}_h^{(0)} = \mathbf{q}_h^n, \\ &i \in [1, \ldots, 5] : \begin{cases} \mathbf{k}^{(i)} = a_i\mathbf{k}^{(i-1)} + \Delta t\mathscr{L}\left(\mathbf{q}_h^{(i-1)}, t^n + c_i\Delta t\right), \\ \mathbf{q}_h^{(i)} = \mathbf{q}_h^{(i-1)} + b_i\mathbf{k}^{(i)}, \end{cases} \\ &\mathbf{q}_h^{n+1} = \mathbf{q}_h^{(5)}, \end{aligned} \tag{21}$$

where $\Delta t = t^{n+1} - t^n$ is the time step size and the coefficients a_i, b_i, c_i are given in Ref. (Carpenter and Kennedy, 1994).

Explicit time-stepping methods impose a conditional stability criterion, $\Delta t \leq C_1/(\max|\lambda_e|)$, where λ_e is the eigenvalue of the spatial discretization and C_1 is a constant related to the size of the stability region of the time-stepping method (Pind et al., 2019). There are two mechanisms at play that influence the maximum allowed time step: (1) the eigenvalue spectra of the matrix operators, and (2) the stiffness of the ADEs. In DGFEM, the matrix operator eigenvalue spectrum scales as $\max|\lambda_e| \sim C_2 c N^{2\gamma}$, where the C_2 constant is dictated by the size of the smallest mesh element and γ is the highest order of differentiation in the governing equations ($\gamma = 1$, here) (Engsig-Karup et al., 2016). Thus, in this work, the time step size is given as

$$\Delta t = C_{\mathrm{CFL}}\min(\Delta x_l)\frac{1}{c}\frac{1}{N^2} \tag{22}$$

where Δx_l is the smallest edge length of the mesh elements and C_{CFL} is a constant of $\mathscr{O}(1)$.

No measures have been taken to automatically address the potential stiffness of the ADEs. Implicit-explicit time-stepping (Kennedy and Carpenter, 2003) or stiffness restriction, by means of a constrained multipole mapping process (Wang et al., 2020), could be used to ensure the ADEs do not cause a numerical instability. However, in all numerical experiments, the spatial scheme is found to determine the condition on the stable time step size.

3.3. Meshing and curvilinear geometry

Meshing of the computational domain Ω_h is done using the open-source meshing tool gmsh (Geuzaine and Remacle, 2009). When affine (straight-sided) elements are used, the mesh generator is used to generate a low-order finite element mesh, and the acoustic simulator then adds the α-optimized nodes to define the high-order mesh elements, in accordance with the polynomial basis order used. To also support handling complex meshes with curvilinear features, curvilinear elements may be introduced. Using curvilinear elements, the internal high-order mesh nodes are shifted to better fit the geometry (Pind et al., 2019). For room acoustic simulations, this is a particularly important feature, since most real-world rooms contain curved or complex shaped boundary surfaces. When using affine elements, the

geometry is described solely by the four vertices of the tetrahedra, but for curvilinear elements the full set of nodes is necessary to describe the geometry. It is therefore of interest to have the mesh generator generate all nodes, and not only the vertices of the low-order finite element mesh as seen for the affine elements.

Currently, gmsh has the ability to generate a high-order curvilinear mesh, however, only with equidistant node distributions. This is sub-optimal, because of the rapid growth of the Lebesque constant associated with equidistant node distributions (Hesthaven, 1998). Our solution here is to apply an interpolation post-processing step on the high-order mesh generated by gmsh, where the equidistant nodes are moved to the α-optimized curvilinear node positions. This is done by re-interpolating the node position by representing the coordinates as polynomial series, cf. (11). Then, let $\mathbf{r}$ and $\mathbf{r}^e$ be the r, s, and t coordinates in the reference element for α-optimized and equidistant node distributions, respectively. Using a hierarchical modal basis, the coefficients $\widehat{\mathbf{r}}$ of this representation can be determined from

$$\mathbf{r} = \mathscr{V}\widehat{\mathbf{r}}, \mathbf{r}^e = \mathscr{V}^{\mathbf{r}^e}\widehat{\mathbf{r}}, \tag{23}$$

where $(\mathscr{V}^{\mathbf{r}^e})_{ij} \equiv \phi_j(\mathbf{r}_i^e)$, using same basis used to define $\mathscr{V}$ but defined in terms of the different set of equidistant nodes. This implies the relationship between the node distributions

$$\mathbf{r} = \mathscr{V}\left(\mathscr{V}^{\mathbf{r}^e}\right)^{-1}\mathbf{r}^e \tag{24}$$

from which an interpolation matrix $\mathscr{I} : \mathbf{r}^e \mapsto \mathbf{r}$ is defined as

$$\mathscr{I} = \mathscr{V}\left(\mathscr{V}^{\mathbf{r}^e}\right)^{-1} \tag{25}$$

which interpolates the equidistant node distribution to the α-optimized node distribution.

4. HPC implementation

The proposed HPC room acoustics simulator is based on the open-source libParanumal framework (Karakus et al., 2019; Świrydowicz et al., 2019), which is a set of highly optimized finite element flow solvers for heterogeneous (GPU/CPU) systems. A Message Passing Interface (MPI) is used to handle communication between CPUs. Note, that when we use the term "CPU," it refers to one single CPU core, while "GPU" refers to the entire GPU. MPI works by treating each CPU core as its own entity, with its own separate memory. Information can be shared between CPU cores through the messaging interface. Also note, that when we use the term "heterogeneous," we refer to the fact that CPUs are utilized for communication and setup; however, the GPUs do all the work during the simulation itself. In practice, this means that libParanumal splits the domain into different chunks. Each CPU core only stores the simulation information needed for the computations in elements belonging to its own part of the domain; however, information has to be shared along interfaces due to the flux.

When executing the code on GPUs, the code is structured such that each CPU core is assumed to have access to one GPU, for example, only CPU_0 can communicate directly with GPU_0. This implies that the interface communications between CPU cores become more costly, as compared to only using CPUs, because CPU_0 must copy all interface information from GPU_0 into RAM on CPU_0, before it can communicate this information to CPU_x through MPI, where $0 < x \leq M - 1$ denotes an arbitrary CPU core out of the total of M CPUs. CPU_x has to copy the received interface information to its connected GPU, GPU_x. In other words, using GPUs introduces an additional communication step, and therefore additional communication overhead, between the CPU and GPU in each time-stepping stage. This implies that we expect a penalty in the scaling of computational performance when adding additional CPU-GPU pairs to run the simulation, as compared to only using CPUs. However, using the GPUs is very attractive because of the very large number of cores available and high on-chip bandwidth, so much faster simulation is expected when using GPUs compared to CPUs. The communication structure can be seen in Figure 1, where the red arrows are the CPU and GPU communications and the blue arrows are the MPI communications.

The implementation of one stage of the time-stepping method in (21), including all the interface communication, is summarized in Algorithm 1. The GPU functions, called

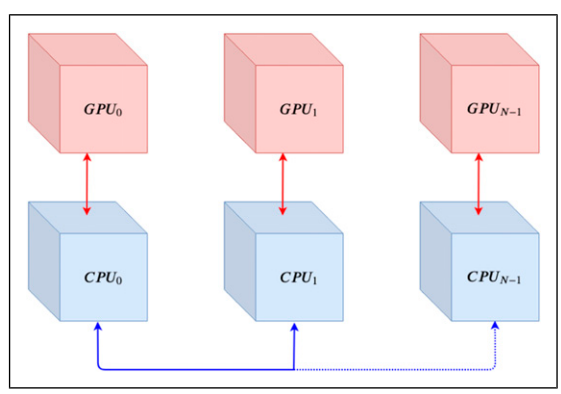

Figure 1. Communication overview of N total heterogeneous CPU-GPU pairs. All CPUs can communicate through MPI, GPUs can only communicate through their attached CPU by proxy.

kernels, must be launched in every iteration of the time-stepping algorithm.

Algorithm 1. One stage of the LSERK time-stepping method.

Input: Previous stage $\mathbf{q}_h^{(i-1)}$
Result: An updated stage $\mathbf{q}_h^{(i)}$
1 GPU to CPU memory copy of interface values (CPU/GPU). Simultaneously initiate volume integral compute kernel (GPU) and non-interface surface inegral compute kernel (GPU).
2 Initiate the interface MPI communication (CPU).
3 CPU to GPU memory copy of interface values (CPU/GPU).
4 Initiate interface surface integral compute kernel (GPU).
5 Update based on current stage (GPU).

On most modern graphics cards the memory movement to/from pinned host memory and device memory can be done with an on-board direct memory access (DMA) engine which can fully overlap with compute activity on the GPU die. This means that the device-host communication, and the volume integral computation and non-interface surface integral computation of the right hand side of (14a)–(14d) can start simultaneously, as shown in step 1. Ideally, the communication work in steps 1, 2, and 3 is finished before the kernels finishes. This way, communication costs are completely hidden. This is possible because the volume kernel and the non-interface surface kernel are completely asynchronous with respect to the host. Step 4, however, cannot start until all communication of steps 1–3 is finished, and the volume kernel is finished.

As can be seen from algorithm 1, the CPU-GPU pairs are currently synchronized in each stage, meaning that an uneven workload distribution will cause a performance penalty. For this reason, it is desirable to have the overall communication work of each CPU equally balanced, as other CPU-GPU pairs will be locked in step 3 until they have sent/received all of the information needed. To ensure load balancing, the mesh is treated as a graph and split among the CPUs using a k-way Recursive Inertial Partitioning.

5. Numerical experiments

All simulations are performed using double-precision floating point numbers. If the numerical accuracy of the double is not required, one can change the datatype to floats. This change will result in a roughly 2× speed increase, and a decrease in the memory footprint by about the same factor. For information about used hardware and compilers, see Appendix B.

5.1. Convergence

5.1.1. Cube domain. As an initial validation of the simulator, a convergence test in a cube domain with $(x, y, z) \in [0, 1]$ and rigid boundaries is carried out. The simulation is initiated using a Gaussian pulse pressure initial condition

$$p(\mathbf{x}, 0) = \exp\left(-\frac{(x-x_s)^2 + (y-y_s)^2 + (z-z_s)^2}{s_{xyz}}\right) \quad (26)$$

where (x_s, y_s, z_s) is the source position and $s_{xyz} = 0.3\ \mathrm{m}^2$ is the spatial variance, which governs the frequency content of the pulse. For this case, an analytic solution exists (Sakamoto, 2007). The domain is meshed using structured meshes of tetrahedral elements. We study both p-convergence, that is, the error behavior as a function of the basis function order for a fixed mesh, and h-convergence, that is, the error behavior as a function of the mesh element size. The numerical error is defined as $e = \|p^{\mathrm{DGFEM}} - p^{\mathrm{TS}}\|_\infty$, where p^{DGFEM} is the simulated pressure in the entire field at $t = 0.01s$ and p^{TS} is the true solution. The time step size is computed using a large time step (close to the stability threshold), by having $C_{\mathrm{CFL}} = 1$ in (22).

Figure 2 shows the resulting p-convergence, tested for three different meshes, having varying resolution. The figure reveals the expected spectral convergence, that is, an exponentially fast decay of errors, is indeed observed.

Figure 3 shows the h-convergence for different orders of the basis. Table 1 lists the resulting convergence rates from a least-squares fitted line. The table also includes convergence rates for a case where the time step size has been reduced significantly ($C_{\mathrm{CFL}} = 0.02$). A convergence rate in the range $O(h^N)$ and $O(h^{N+1})$ is expected (Hesthaven and Warburton, 2008). The results are mostly in good agreement with the expected rates, with the exception of $N = 1$ and $N = 2$. This is explained by the meshes not being fine enough for these low orders to give an accurate representation of their convergence. Furthermore, it is seen that reducing the time step size has only marginal influence on the convergence rates, indicating that in this test case, the spatial errors dominate.

5.1.2. Cylinder domain. Consider now a cylinder shaped domain, of 1 m height and 1 m radius and rigid boundaries. A convergence test is carried out, where the domain is

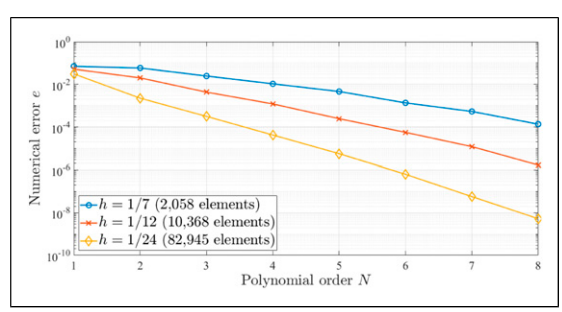

Figure 2. Semi-log plot showing the p-convergence for the rigid cube test case.

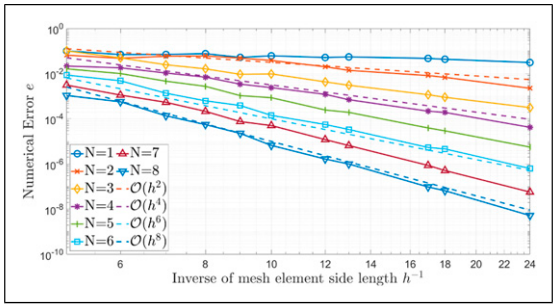

Figure 3. Log–log plot showing the *h*-convergence for the rigid cube test case.

Table 1. Convergence rates for the *h*-convergence study.

N	$C_{\mathrm{CFL}} = 1$	$C_{\mathrm{CFL}} = 0.02$
1	−0.778	−0.778
2	−3.239	−3.240
3	−3.919	−3.920
4	−4.618	−4.620
5	−5.752	−5.752
6	−6.188	−6.188
7	−7.754	−7.747
8	−8.161	−8.166

meshed either using affine mesh elements or curvilinear mesh elements. The reference solution here is computed using an extremely fine mesh of affine elements (410,308 elements, $h \approx 0.06$) and $N = 8$ basis function order. The numerical error is then defined as $= \left\| p^{\mathrm{DGFEM}} - p^{\mathrm{DGFEM}}_{\mathrm{fine}} \right\|$, where p^{DGFEM} is the simulated solution in 10 randomly chosen receiver points at time $t = 0.01$ s and $p^{\mathrm{DGFEM}}_{\mathrm{fine}}$ is the simulated solution in the same receiver points for the fine mesh reference simulation. All simulations have been run with $C_{\mathrm{CFL}} = 0.02$ to ensure that spatial errors dominate.

Figure 4 shows the resulting *p*-convergence, tested for three different meshes of varying resolution, using either affine elements or curvilinear elements. When using affine elements, the high-order convergence is destroyed, as the convergence rate flattens out for orders higher than $N \approx 3$. However, when using the curvilinear elements, spectral convergence is observed.

Further insights are offered in Figure 5, where *h*-convergence is shown for both the affine and the curvilinear case. It is seen that for the affine elements, a convergence rate of around $O(h^2)$ is achieved consistent with the geometric approximation, whereas a much higher convergence rate is observed for the curvilinear elements. However, the error plateaus at around 10^{-7}, which is likely due to aliasing errors that stem from the transformation Jacobians and normals no longer being constant, which means the integrands are no longer purely polynomial. This results in inexact integration of variable terms in the inner products used to compute the matrix based operators. Anti-aliasing techniques could be employed to reduce the error further (Engsig-Karup et al., 2016; Kirby and Karniadakis, 2003) or the use of cubature formulas to evaluate the inner products (Hesthaven and Warburton, 2008). However, in practical room acoustics simulations, error levels below 10^{-7} are rarely needed, with the actual error requirements being around 5×10^{-3} to 2×10^{-2} (Sampedro Llopis et al. (2022)), and, furthermore, such techniques increase the computational cost of the scheme.

5.2. Boundary condition validation

A validation study of the locally reacting frequency dependent boundary conditions is carried out, using a normal incidence single reflection case. For this case, an analytic solution exists (Thomasson, 1976). A 6 m × 10 m × 10 m rectangular domain is meshed with an unstructured mesh of tetrahedral elements. The frequency dependent impedance boundary is the *YZ* plane at $x = 0$. The resolution of the mesh is roughly 8 points per wavelength (PPW) at 1 kHz, the highest frequency of interest. The source is placed 2 m from the impedance boundary (*S*: (2, 5, 5) m) and the receiver is placed between the source and the boundary (*R*: (1, 5, 5) m). The simulation duration is $t_{\mathrm{final}} = 0.02$ s, which ensures that no parasitic reflections from the other room surfaces arrive at the receiver position. A Gaussian pulse initial condition, centered at the source position, with $s_{xyz} = 0.2$ m^2 is used.

Two boundary conditions are considered; a porous material mounted on a rigid backing (BC1) and a thin porous layer backed by an air cavity (BC2). The surface admittance of the BCs is estimated using Miki's model and a transfer matrix method (Allard and Atalla, 2009; Miki, 1990). Vector fitting is used for the multipole mapping (Gustavsen and Semlyen, 1999). The material is mapped in the frequency range of 50 to 1500 Hz. An estimate of the relative mapping error is given by

$$\epsilon = \frac{\overline{\left| Y(f) - Y_{\mathrm{fit}}(f) \right|}}{\overline{Y}(f)} \tag{27}$$

where $\overline{Y}$ indicates the mean across frequency. Table 2 summarizes the properties of the considered boundary conditions and the multipole mapping. Note that more poles are needed in the mapping of BC1, because the admittance response fluctuates more with frequency for this absorber.

Figure 6 shows the resulting Fourier transformed pressure responses of the single reflection study. An excellent match between the analytic solution and the simulation is found for both BCs considered, both in terms of magnitude and phase. This confirms the high precision of the BC implementation.

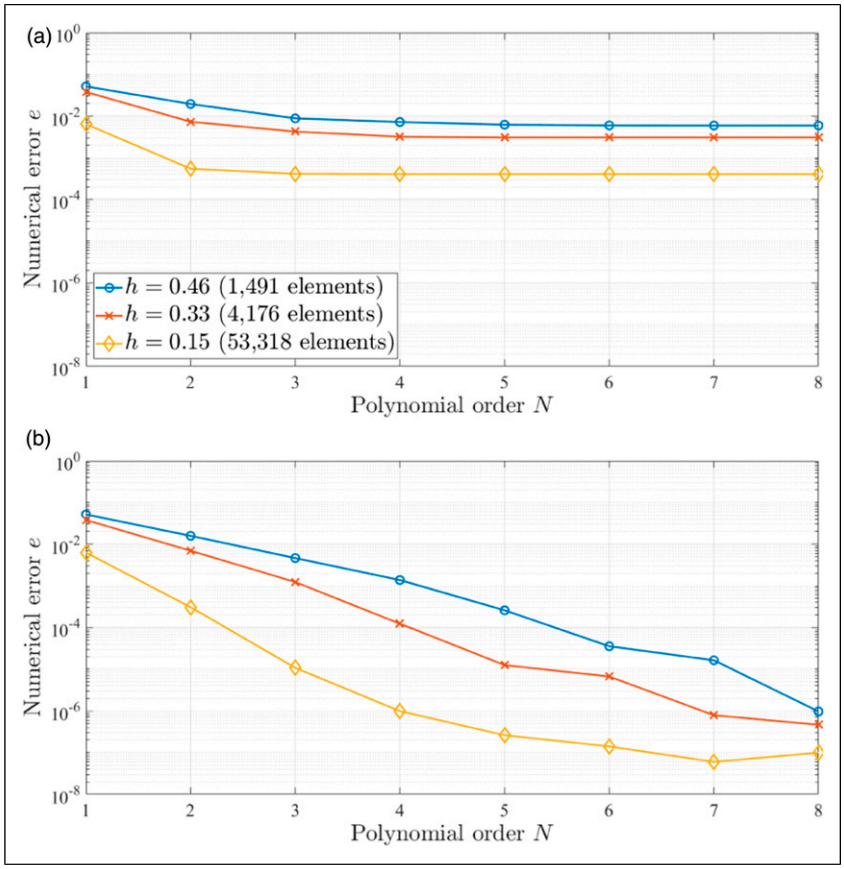

Figure 4. Semi-log plots showing the *p*-convergence for the cylinder geometry using either affine or curvilinear meshes. (a) Affine elements. (b) Curvilinear elements.

5.3. Heterogeneous multi-device CPU-GPU performance

5.3.1. Strong scaling. Strong scaling is characterized by how the runtime changes while keeping the problem size constant and increasing the number of processing units. Thus the overall workload is constant, while the workload of each processing unit decreases, as the workload is split among an increasing number of processing units.

Six meshes of varying resolutions are constructed, described in Table 3. These meshes are used to analyze the strong scaling of the simulator for two different polynomial orders, $N = 4$ and $N = 6$. In all cases, rigid BCs are used.

A measure of the overall strong scaling computational efficiency of the simulator is defined as

$$\eta_S = \frac{T_R^b M^b}{T_R M} \tag{28}$$

where T_R^b denotes the runtime of a baseline case, M^b denotes the number of GPUs used for the baseline case, and T_R and M denote the same for the comparison case. Thus, if $\eta_S = 1$, perfect scaling occurs when going from M^b to M.

Before the strong scaling tests, we show the achieved performance on mesh 4 for both $N = 4$ and $N = 6$. This is done with a roofline analysis using the NVIDIA Nsight Compute tool on a single Nvidia A100 GPU, and can be seen in Figure 7. Both cases appear to be compute bound in the volume kernel, with the surface kernel having a slightly lower performance. Note that in general more time is spent in the volume kernel than in the surface kernel, however, how much more time is spent, is highly dependent on both mesh geometry and polynomial order.

An array of simulations is carried out, using the meshes defined in Table 3. To account for noise, the runtime is defined as the average runtime of three simulations. The resulting strong scaling computational efficiency is shown in Table 4, with the corresponding runtimes per time step available in Appendix C.1.

As expected, the scaling is dependent on whether the problem size is large enough to hide communication. While every mesh scales relatively well to two GPUs, we start to see the scaling deteriorate for the smaller meshes once the GPU count is increased. However for the larger meshes, especially mesh 4 and 5, we see great scaling for all GPU setups. To better visualize the scaling, Figure 8 shows the

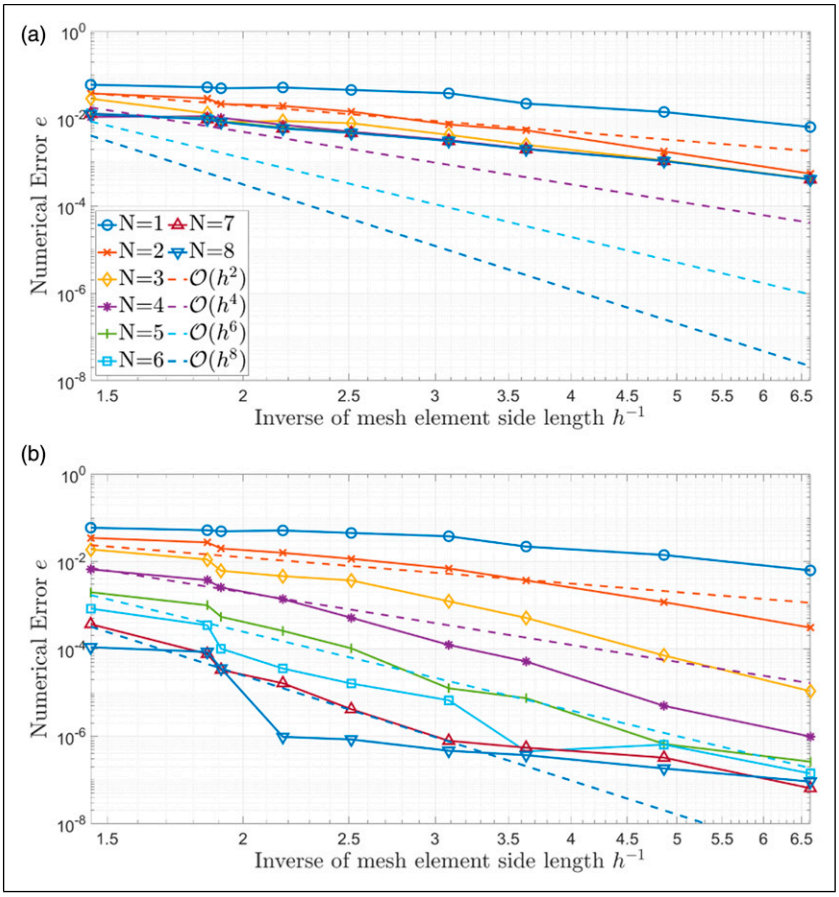

Figure 5. Log–log plots showing the h convergence for the cylinder geometry using either affine or curvilinear meshes. (a) Affine elements. (b) Curvilinear elements.

Table 2. Material properties and multipole mapping error for the BCs used in the boundary validation study. d_{mat} [m] is the porous material thickness, d_0 [m] is the air cavity depth, σ_{mat} [Ns/m^4] is the flow resistivity of the porous material, N_{poles} is the number of poles used in the multipole mapping, and ϵ_{Re} [%] and ϵ_{Im} [%] are the relative mapping errors of the real and imaginary parts of the surface admittance, respectively.

ID	d_{mat}	d_0	σ_{mat}	N_{poles}	ϵ_{Re}	ϵ_{Im}
BC1	0.02	0.2	50,000	8	0.0216	0.0177
BC2	0.05	0.0	10,000	4	0.0213	0.0071

speed-up factors. Here it is clear that smaller meshes stagnate when the GPU count is increased. This is to be expected as both of these meshes are relatively small, leading the communication time to dominate the runtime in these cases.

We now repeat the same tests, but using a polynomial order $N = 6$ basis. This increases the number of DoFs involved. Mesh 5 is not considered due to memory limitation on a singly GPU. Table 5 shows the strong scaling efficiency factor η_S, with the corresponding runtimes per time step available in Appendix C.2. As expected, the scaling efficiency has improved, as compared to the $N = 4$ case, due to the increase in DoF.

To visualize this, we combine the information about mesh 1 through 4 from Tables 4 and 5 in Figure 9. Here we clearly see the efficiency increase when comparing $N = 4$ to $N = 6$ for all shown meshes.

In general, strong scaling is hampered by a number of challenges, which cause the loss in efficiency. Latency accumulates for copying kernel arguments from host to device, for launching a kernel, for copying halo data between and device, and MPI message passing latency in the halo exchange. Launching each kernel incurs a time overhead that is fixed relative to the problem size. On a NVIDIA V100 the latency can be as low as 5 μs but as high as 20 μs if kernels are not adequately pipelined. We apply an Amdahl type scaling model adapted for a memory bound kernel that streams data between the GPU cores and the GPU device memory. To achieve 80% efficiency in streaming data, the kernel must stream at least $B = 4R_\infty T_0$ bytes of data, where R_∞ is the maximum streaming rate of

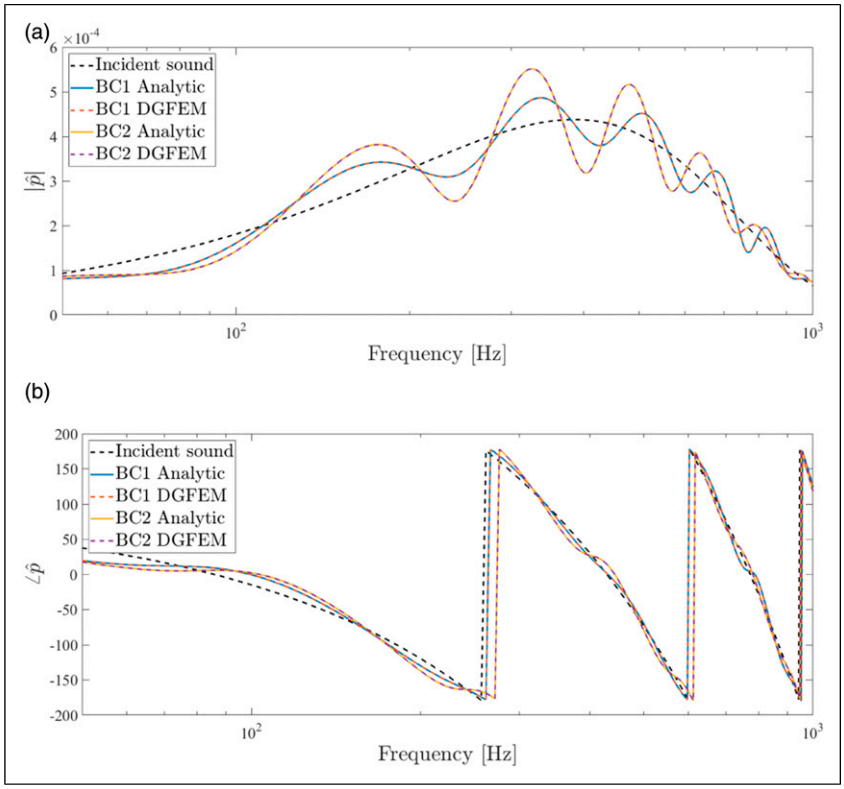

Figure 6. Simulation of a single reflection from a locally reacting frequency dependent impedance boundary compared against the analytic solution. (a) Magnitude. (b) Phase.

Table 3. Meshes and resulting DoF used for strong scaling analysis, where k denotes kilo, that is, × 1000.

	#Elements	DoF N = 4	DoF N = 6
Mesh 1	50k	7000k	16,800k
Mesh 2	200k	28,000k	67,200k
Mesh 3	500k	70,000k	168,000k
Mesh 4	1000k	140,000k	336,000k
Mesh 5	3000k	420,000k	1,008,000k

Table 4. Strong scaling computational efficiency η_S for the meshes in Table 3 with N = 4.

#GPUs	Mesh 1	Mesh 2	Mesh 3	Mesh 4	Mesh 5
1	1	1	1	1	1
2	0.8929	0.977	0.9790	0.9874	0.9944
4	0.4425	0.8426	0.9709	0.9775	0.9923
8	0.2671	0.4851	0.9392	0.9600	0.9862
12	0.1809	0.3151	0.7880	0.9416	0.9799

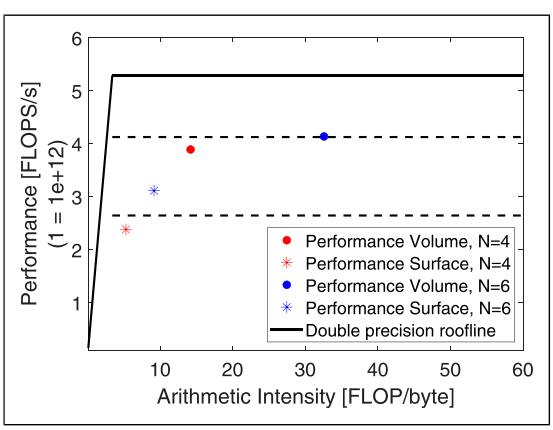

Figure 7. Roofline test for mesh 4 for both N = 4 and N = 6. Dashed lines represent 50% and 80% of roofline.

the kernel and T_0 is the time it takes to launch the kernel. If the kernel is to achieve 80% of peak throughput it would need to stream approximately $B = 4 \times 900$ GB/s $\times$ 5μs $\approx$ 18 MB. Assuming that the kernel only streams double precision variables, then we see that at least two million field values would have to be streamed to and/or from device memory to achieve the target 80% efficiency of the GPU memory bus. Through the lens of this simplified model, it is apparent that the efficiency of a memory bound kernel degrades for small problem sizes which is precisely the regime we enter when performing a strong scaling study, wherein the number of GPUs is increased but the amount of problem data remains fixed. In practice the kernel launch time can be significantly higher and the achieved throughput

can be significantly lower, however the fundamental limit to strong scaling remains.

5.3.2. Weak scaling. Weak scaling is characterized by how the runtime changes when fixing the problem size for each processing unit, that is, the overall workload is constant on

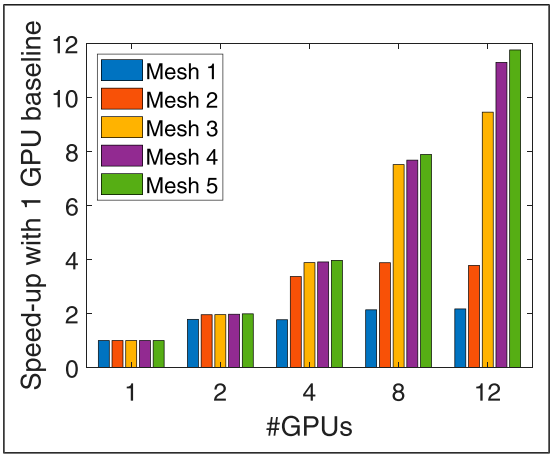

Figure 8. Speed-up factors, with 1 GPU configuration as a baseline.

Table 5. Strong scaling computational efficiency η_S for meshes 1 through 4 in Table 3 with $N = 6$.

#GPUs	Mesh 1	Mesh 2	Mesh 3	Mesh 4
1	1	1	1	1
2	0.9516	0.9897	0.9998	0.9885
4	0.6844	0.9723	0.9935	0.9942
8	0.4317	0.7510	0.9751	0.9915
12	0.3059	0.4863	0.9672	0.9802

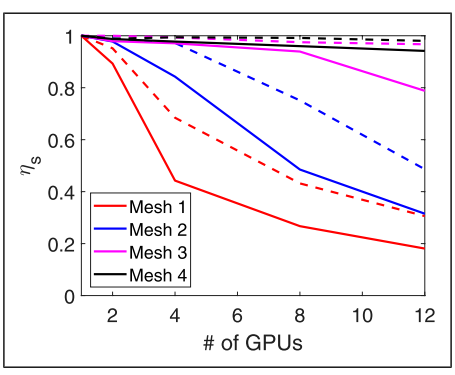

Figure 9. Strong efficiency for mesh 1 through 4. Full lines are $N = 4$, and dashed lines are $N = 6$.

Table 6. Meshes and resulting DoF used for weak scaling analysis, where k denotes kilo, that is, × 1000.

	#Elements per GPU	DoF $N = 4$	DoF $N = 6$	DoF $N = 8$
Series 1	50k	7000k	16,800k	33,000k
Series 2	200k	28,000k	67,200k	132,000k

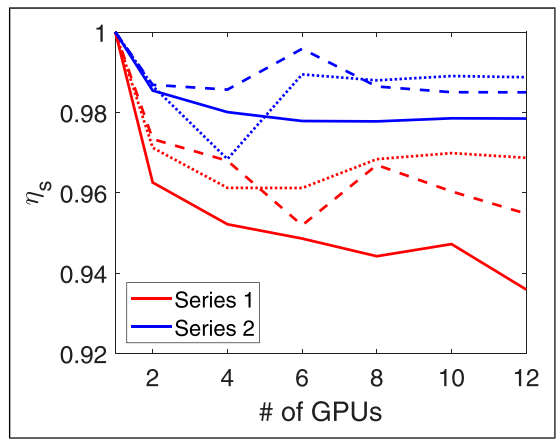

Figure 10. Weak scaling computational efficiency for the meshes in Table 6. Full lines are $N = 4$, dashed lines are $N = 6$, and dotted lines are $N = 8$.

Table 7. Strong scaling computational efficiency η_S for different boundary conditions.

	Mesh 1			Mesh 2		
#GPUs	Rigid	BC1	BC2	Rigid	BC1	BC2
1	1	1	1	1	1	1
2	0.9516	0.9516	0.949	0.9885	0.9967	0.989
4	0.6844	0.7181	0.720	0.9942	0.9959	0.984
8	0.3059	0.4355	0.413	0.9915	0.9874	0.977

Table 8. Relative change in runtime compared to the rigid case.

	Mesh 1		Mesh 2	
#GPUs	BC1, %	BC2, %	BC1, %	BC2, %
1	+11.03	+3.34	+7.74	+1.54
2	+11.03	+4.66	+7.93	+1.47
4	+5.82	−0.017	+7.55	+2.61
8	+10.05	+4.22	+8.19	+3.08

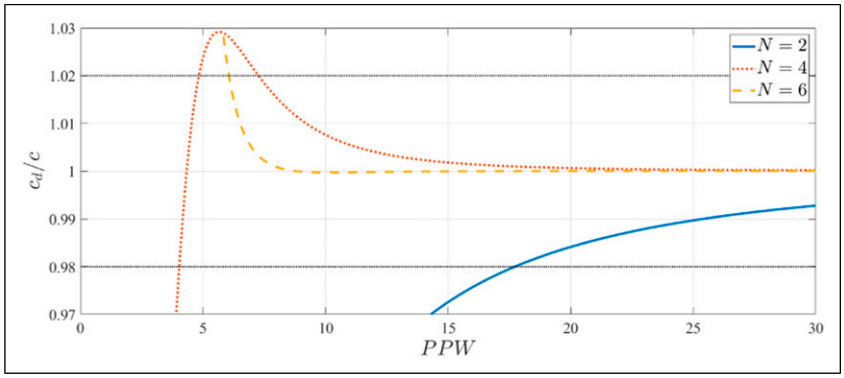

Figure 11. 1D dispersion analysis results when using Δt = 0.0001, c = 1 m/s, and n = 2 time steps.

each processing unit. The weak scaling computational efficiency is given by

$$\eta_W = \frac{T_R^b}{T_R} \tag{29}$$

As for strong scaling, runtime is taken as the average runtime of three runs, to account for noise. To show the weak scaling, we construct two series of meshes with 50 k and 200 k elements per GPU, respectively. Additional mesh information is shown in Table 6.

Figure 10 shows great weak scaling for all cases. We also generally see an increase in scaling efficiency when increasing the polynomial order, though for both series, N = 6 and N = 8 are relatively close in performance.

The results for both the strong and weak scaling show that the number of GPUs one should use depends heavily on the problem size. Adding more compute power for an application can greatly decrease the overall runtime until some threshold is reached, where the communication overhead begins dominating the runtime. Importantly, this implies that if the number of GPUs used for solving a problem is selected in such a way that each GPU has a high workload, an excellent scaling can be achieved to large numbers of GPUs, thus allowing the efficient simulation of very large problems.

5.3.3. Influence of BCs on performance. By having non-rigid BCs, an additional computation must take place at the domain boundary nodes, particularly for the case of frequency dependent BCs, where the ODE system in (9) must be solved at the boundary. An experiment is carried out, where the influence of the non-rigid BCs on runtime and on strong scaling is investigated. For this experiment, we consider meshes 1 and 4 in Table 3 with N = 6. These are cube shaped domains (all bounding surfaces are equally large). We set the BCs such that all surfaces are frequency dependent impedance boundaries, with the BCs defined in Table 2.

Table 9. Points-per-wavelength needed to ensure spatial dispersion errors remain under 2%, based on the 1D dispersion analysis of Figure 11.

	N = 2	N = 4	N = 6
Points per wavelength	17.7	7.3	6.0

The strong scaling results are shown in Table 7 and the relative change in runtime when having non-rigid BCs, as compared to the rigid case, is shown in Table 8. The runtime increases by an average of 8.67% and 2.61% for BC1 and BC2, respectively. A difference between the effect of BC1 and BC2 is to be expected due to the difference in the amount of poles each BC requires for computation. The strong scaling results also show that scaling is not influenced by the use of realistic boundary conditions, as in all cases the scaling matches up with the original rigid boundary condition test. These results show that having realistic boundary conditions only has a marginal influence on the performance. The runtime is more affected for the smaller mesh – this is expected since the boundaries are a larger proportion of this mesh. Mesh 1 has roughly 40% more boundary faces per element in comparison to mesh 4. This matches relatively well with the difference in runtimes, when comparing the increase in runtime in comparison to the rigid case. BC1 has a $\sim 23\%$, and BC2 a $\sim 40\%$ larger increase in runtime on mesh 1, compared to the increase in runtime on mesh 2.

5.4. Dispersion and runtime as a function of simulation frequency range

In practice, it is desirable to simulate a large portion of the audible frequency spectrum with the wave-based simulator and therefore of interest to analyze how this affects the runtime. The valid frequency range of the simulation depends on the spatio-temporal resolution—a high enough

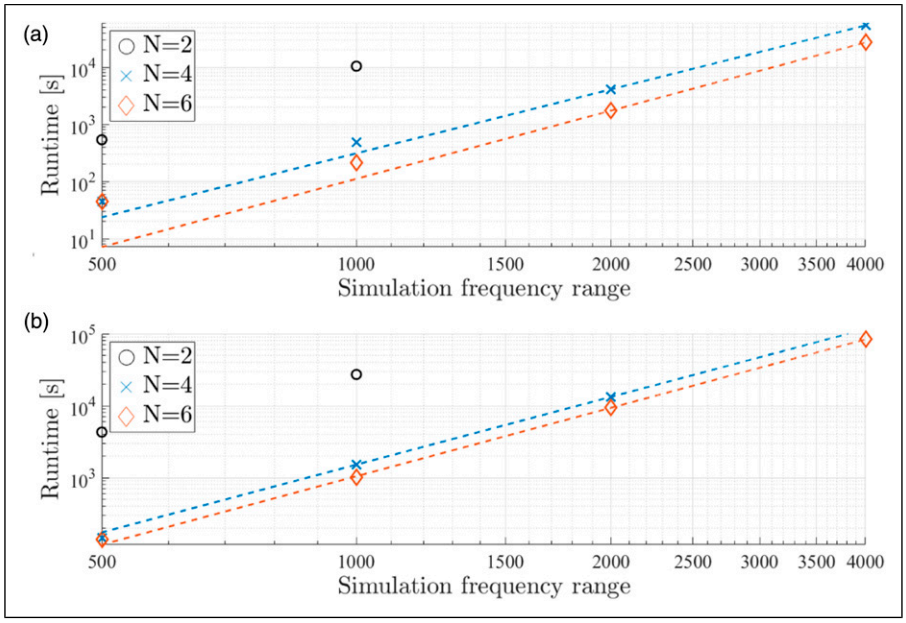

Figure 12. Measured runtime as a function of simulation frequency range f_{max} when using Δt = 0.0001. The dashed lines are the fitted runtime model on the form $r = af_{max}^4$. (a) Bedroom. (b) Classroom.

resolution must be chosen such that numerical errors are below a certain threshold, generally taken to be around 2% (Saarelma et al., 2016).

Using high-order basis functions will result in improved accuracy of the simulation for a given mesh resolution, which in turn means that fewer PPW can be used to achieve a certain level of accuracy (Kreiss and Oliger, 1972; Pind et al., 2019). However, this will also affect performance due to a reduction in time step size as per equation (22). A 1D dispersion analysis of a single wave mode propagation is carried out to roughly assess the PPW needed for different basis function orders to maintain numerical dispersion error below the audibility threshold. The 1D advection equation is

$$u_t + cu_x = 0 \tag{30}$$

which has solutions on the form $u(x, t) = f(kx - \omega t) = f((\omega/c)x - \omega t)$, where $f(s)$ is the initial condition. By assuming a solution ansatz $f(s) = e^{js}$, the exact solution after n time steps will have a phase shift corresponding to $e^{-\omega n \Delta t}$. Thus, the numerical solution at time step n, u^n, and the initial condition u^0, are related by

$$u^0 = u^n e^{-\widehat{\omega} n \Delta t} \tag{31}$$

where $\widehat{\omega}$ is the numerical frequency, which will differ from the exact frequency ω due to the dispersion of the numerical scheme. This non-linear equation can be solved numerically for $\widehat{\omega}$, and in this work, a Levenberg–Marquardt algorithm is used. By comparing the numerical frequency against the exact one, the dispersion relationship can be established since $c_d/c = \widehat{\omega}/\omega$, where c_d is the numerical wave speed.

Figure 11 shows an example dispersion relationship for the $N = 2, 4, 6$ basis function orders. Note that this is only intended as a rough estimate of the PPW needed to maintain the desirable error levels, as in practice the

Table 10. Coefficients a for the runtime model $r = af_{max}^4$ and associated R^2 values for evaluating the goodness of the fit.

	Bedroom		Classroom	
N	a	R^2	a	R^2
2	1.04×10^{-8}	0.92	2.74×10^{-8}	0.96
4	2.11×10^{-10}	0.86	8.36×10^{-10}	0.91
6	1.06×10^{-10}	0.85	3.28×10^{-10}	0.88

Table 11. Details regarding the rooms and the meshes used.

Room	Auditorium	Concert hall
Dimensions	10 m × 10 m × 50 m	10 m × 40 m × 50 m
Volume	5000 m^3	20,000 m^3
#Elements	1920k	7680k
DoF	268,800k	1,075,200k
Δt	2.94×10^{-5}	2.94×10^{-5}
Schröder freq.	28 Hz	20 Hz

Table 12. Runtimes in seconds for the large shoebox-shaped rooms.

#GPUs	Auditorium	Per time step	Concert hall	Per time step
1	208.31	0.3063	—	—
2	104.95	0.1543	—	—
4	52.97	0.1543	209.79	0.3085
8	26.88	0.0395	105.65	0.1554
12	21.91	0.0322	71.31	0.1049

dispersion will also be dependent on simulation duration and direction.

Table 9 lists the PPW needed to ensure that spatial dispersion errors remain under 2% for the example case. Using these PPW values as reference, we can construct meshes with the right resolution to ensure valid simulation results up to a certain frequency $f_{\max}$. This allows us to analyze how the runtime changes as the frequency range of the simulation is extended, when using different basis function orders. Two rooms are considered, a 4 m × 2.7 m × 3 m "bedroom" and a 9 m × 7 m × 3 m "classroom." In both cases, frequency independent BCs are used and a simulation time (impulse response length) of 1 s is simulated. The rooms are simulated using 4 GPUs.

Figure 12 shows the measured runtime. The runtime is measured for $f_{\max}$ = 250, 500, 1000, 2000, 4000 Hz. The runtime increases quickly with frequency in all cases, after a certain frequency is reached. This is due to the inherent nature of having to mesh the three dimensional volume of the room. However, the runtime starts to increase rapidly at a much higher frequency for the higher order cases. For $N = 2$ we did not go beyond $f_{\max}$ = 1000 Hz due to computational cost. A function on the form $r = af_{\max}^4$ is fitted to the measured runtimes, where r is the estimated runtime. As can be seen in Table 10, a is considerably lower when using the higher order basis functions.

5.5. Large shoebox-shaped rooms

To give additional insights into the performance of the simulator for challenging practical cases, we consider two very large shoebox-shaped rooms. The first is 5000 m^3, which is meant to imitate a large auditorium, and the other is 20,000 m^3, which is meant to imitate a large concert hall. These rooms are meshed using a structured mesh, $N = 4$ basis function order and the spatial resolution is 6 PPW at 1000 Hz. In practice, a common rule of thumb for second order schemes is to use 6 PPW at the highest frequency of interest. Thus, it is reasonable to assume that when using $N = 4$ basis order, the accuracy is likely acceptable at frequencies beyond 1000 Hz. For reference, the Schroeder frequency of these rooms is 28 Hz and 20 Hz, for the auditorium and the concert hall, respectively, assuming the auditorium has a reverberation time of 1 s and the concert hall a reverberation time of 2 s. The room and mesh details are summarized in Table 11.

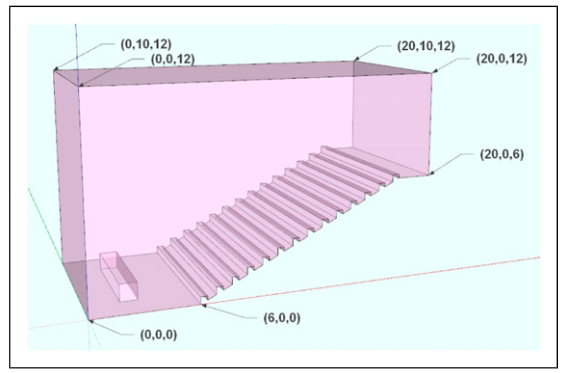

Figure 13. Geometric model of auditorium B341-A21 at DTU.

These rooms are simulated for a time of 2 s for a total of 680 time steps. Boundary conditions are rigid. Table 12 lists the resulting total runtimes and runtimes per time step.

For the concert hall, the simulation cannot be run on 1 or 2 GPUs due to memory limitations, but for both the auditorium and the concert hall, the runtimes improves significantly when increasing the amount of GPUs, which aligns with our earlier scaling tests. While the auditorium has slightly stagnated in scaling when going to 12 GPUs, the trend of the concert hall indicates that the runtimes would likely drop further if more than 12 GPUs were used. We have thus shown that for very large rooms, we can simulate up to 30 times the Schroeder frequency within practical timeframes. However, one has to keep in mind that this simulation was run on structured meshes in a rectangular room with rigid BCs. A more complex geometry, which has to be meshed as an unstructured mesh, could lead to smaller time steps and therefore longer runtime.

5.6. Complex geometry cases

5.6.1. Auditorum B341-A21 at DTU. As an example of using the simulator on realistic 3D geometries, consider the

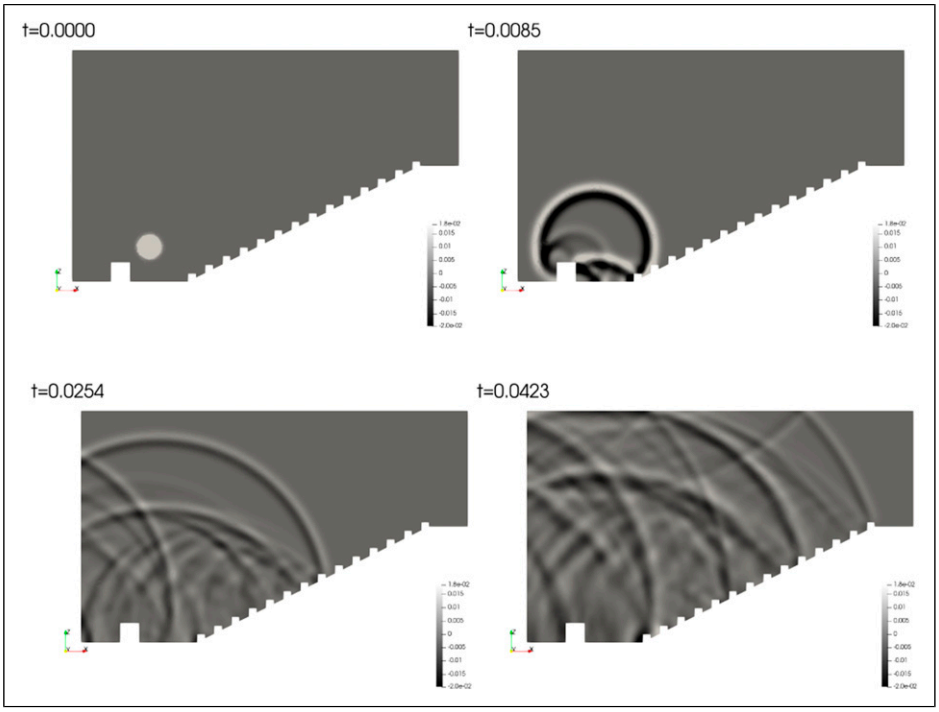

Figure 14. Snapshots of the acoustic wave propagating inside the auditorium.

room in Figure 13, which shows a simplified model of auditorium B341-A21 at the Technical University of Denmark (DTU). As can be seen, the model contains a desk at the bottom and several benches on the slope. Boundary conditions have been chosen to represent a realistic scenario, where the floor, table, and benches are set to be rigid, the walls have frequency independent impedance boundary conditions with $Z = 15{,}630$ Ns/m^4, which corresponds to approximately 10% absorption at normal incidence, and the ceiling uses frequency dependent boundary conditions. The ceiling is modeled as a suspended porous material, with thickness $d_{\text{mat}} = 0.02$ m, and air cavity depth of $d_0 = 0.2$ m and having flow resistivity $\sigma_{\text{mat}} = 50{,}000$ Ns/m^4. The initial condition is a Gaussian pulse with width $s_{xyz} = 0.3$ m^2, placed at $S = (4.0, 5.0, 1.8)$ m, to mimic the position of a lecturer.

Figure 14 presents a series of snapshots of the solution in the XZ-plane at $y = 5$ m, showing the propagation of the acoustic wave within the domain. Note that the diffraction patterns from the benches and the desk are being accurately captured.

5.6.2. Star-shaped room with a dome. As a last test case, consider the geometry shown in Figure 15, which consists of three overlapping rectangles and a dome shaped roof in the center. This geometry is meshed with curvilinear elements to accurately capture the curved shape of the dome. The geometry is similar to what is used in a work by Bilbao (Bilbao, 2013); however, in that work, the geometry is meshed with affine elements and simulated with a second-order accurate method. The floor is made to be highly absorptive, having a normal incidence surface impedance of $Z = \rho c = 411.6$ Ns/m^4, while other surfaces are perfectly reflecting. Figure 16 shows snapshots of the wave propagation in the room, both in a section view and a plan view. Note the acoustic focusing effect due to the dome.

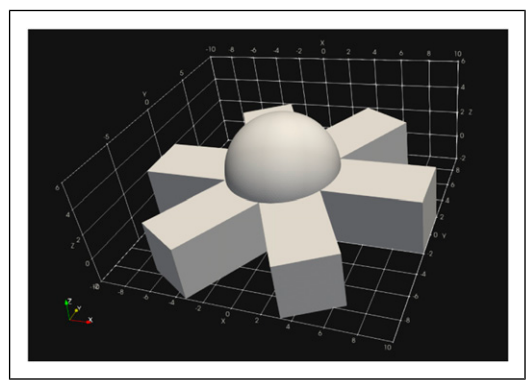

Figure 15. Geometric model of dome shaped room.

The two complex geometry cases presented in this section and the previous one show that the simulator is capable of using the geometrical flexibility of the

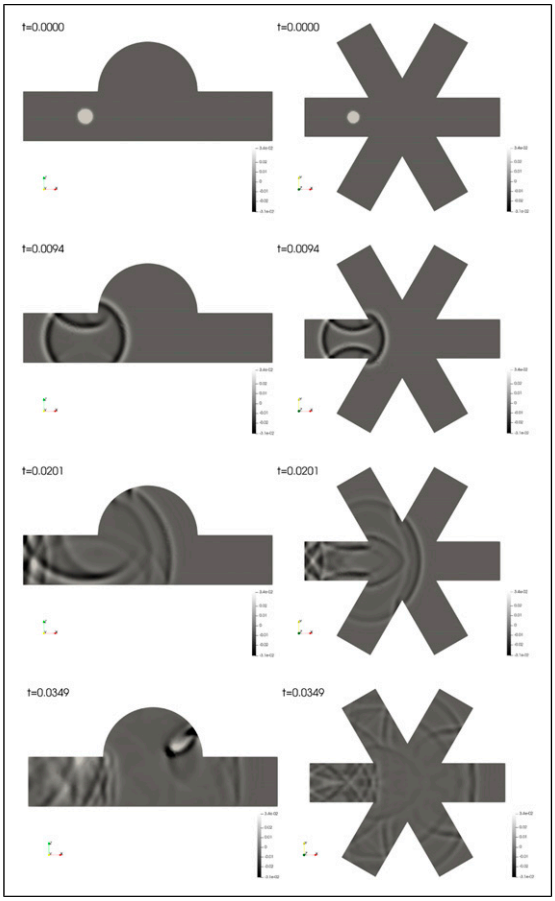

Figure 16. Snapshots of the acoustic wave propagating inside the star-shaped room.

DGFEM method to simulate realistic large-scale geometries.

6. Conclusions

A massively parallel nodal DGFEM-based room acoustic simulator has been presented and its performance analyzed. It has been shown that the simulator can handle large and complex problems, including curvilinear geometries and frequency dependent boundaries, under short runtimes. This makes the simulator suited for use on problems of industrial size and complexity. The work presented here extends the usability of wave-based room acoustic simulations far beyond the restricted use-case of small rooms and frequencies below the Schroeder frequency, which wave-based methods are historically typically associated with.

The heterogeneous multi-device scalability benchmarks show that the simulator can handle problems of nearly arbitrary size, as long as the number of GPUs is selected to ensure that the workload on each GPU is sufficient, to avoid costs, related to communication between CPU cores starting to dominate the runtime. While the new model has been designed for massive parallelism and scalability utilizing heterogeneous multi-device CPU-GPU many-core hardware, there are still opportunities for further improvements of the performance. It is also shown that using mesh-adaptive high-order numerical methods is highly beneficial for room acoustic simulations, due to the improved physical description such wave-based room acoustic model offer.

Acknowledgments

The authors wish to thank Mr Hermes Sampedro Llopis and Mr Nikolas Borrel-Jensen for fruitful discussions.

Declaration of conflicting interests

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Funding

The author(s) disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: This work was supported by the This work is partly supported by HPC innovation for European SMEs (FF4EuropHPC), Grant ID 951745, European High-Performance Computing Joint Undertaking, European Community Horizon 2020 Programme. The work was also supported by DTU Computing Center (DCC) with access to state-of-the-art computing resources.

ORCID iDs

Anders Melander https://orcid.org/0000-0003-3310-9638
Cheol-Ho Jeong https://orcid.org/0000-0002-9864-7317
Noel Chalmers https://orcid.org/0000-0002-1293-7525

References

Ainsworth M (2004) Dispersive and dissipative behaviour of high order discontinuous Galerkin finite element methods. *Journal of Computational Physics* 198(1): 106–130.

Allard JF and Atalla N (2009) *Propagation of Sound in Porous Media: Modelling Sound Absorbing Materials*. 3rd edition. West Sussex, UK: Wiley, Ch. 3, 5.

Allen JB and Berkley DA (1979) Image method for efficiently simulating small-room acoustics. *Journal of the Acoustical Society of America* 65: 943–950.

Bilbao S (2013) Modeling of complex geometries and boundary conditions in finite difference/finite volume time domain room acoustics simulation. *IEEE Transactions on Audio Speech and Language Processing* 21(7): 1524–1533.

Borrel-Jensen N, Engsig-Karup AP and Jeong C-H (2021) Physics-informed neural networks for one-dimensional sound field predictions with parameterized sources and impedance boundaries. *JASA Express Letters* 1: 122402.

Botteldooren D (1995) Finite-difference time-domain simulation of low-frequency room acoustic problems. *Journal of the Acoustical Society of America* 98(6): 3302–3308.

Carpenter MH and Kennedy C (1994) Fourth-order 2N-storage Runge-Kutta schemes. *NASA Report TM 109112*. Hampton, VA: NASA Langley Research Center.

Craggs A (1994) A finite element method for the free vibration of air in ducts and rooms with absorbing walls. *Journal of Sound and Vibration* 173(4): 568–576.

Dragna D, Pineau P and Blanc-Benon P (2015) A generalized recursive convolution method for time-domain propagation in porous media. *Journal of the Acoustical Society of America* 138(2): 1030–1042.

Eleuterio FT (1999) *Riemann Solvers and Numerical Methods for Fluid Dynamics: A Practical Introduction*. 2rd edition. Berlin, Germany: Springer-Verlag.

Engsig-Karup AP, Eskilsson C and Bigoni D (2016) A stabilised nodal spectral element method for fully nonlinear water waves. *Journal of Computational Physics* 318: 1–21.

Geuzaine C and Remacle J-F (2009) Gmsh: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities, *International Journal for Numerical Methods in Engineering* 79(11): 1309–1331.

Gustavsen B and Semlyen A (1999) Rational approximation of frequency domain responses by vector fitting, *IEEE Transactions on Power Delivery* 14(3): 1052–1061.

Hargreaves JA, Rendell LR and Lam YW (2019) A framework for auralization of boundary element method simulations including source and receiver directivity, *Journal of the Acoustical Society of America* 145(4): 2625–2637.

Hesthaven J (1998) From electrostatics to almost optimal nodal sets for polynomial interpolation in a simplex. *SIAM Journal on Numerical Analysis* 35(2): 655–676.

Hesthaven JS and Warburton T (2008) *Nodal Discontinuous Galerkin Methods – Algorithms, Analysis, and Applications*. New York, NY: Springer.

Hornikx M, Krijnen T and van Harten L (2016) openPSTD: the open source pseudospectral time-domain method for acoustic propagation. *Computer Physics Communications* 203: 298–308.

Hu F and Atkins H (2002) Two-dimensional wave analysis of the discontinuous Galerkin method with non-uniform grids and boundary conditions. In: 8th AIAA/CEAS aeroacoustics conference and exhibit, Breckenridge, Colorado, 17–19 June 2002.

Karakus A, Chalmers N, Świrydowicz K, et al. (2019) A GPU accelerated discontinuous Galerkin incompressible flow solver. *Journal of Computational Physics* 390: 280–404.

Karniadakis GE, Kevrekidis IG, Lu L, et al. (2021) Physics-informed machine learning. *Nature Reviews Physics* 3: 422–440.

Kennedy CA and Carpenter MH (2003) Additive Runge-Kutta schemes for convection-diffusion-reaction equations. *Applied Numerical Mathematics* 44(1): 139–181.

Kirby RM and Karniadakis GE (2003) De-aliasing on non-uniform grids: algorithms and applications, *Journal of Computational Physics* 191(1): 249–264.

Kreiss H-O and Oliger J (1972) Comparison of accurate methods for the integration of hyperbolic equations. *Tellus* 24(3): 199–215.

Krokstad A, Strom S and Sørsdal S (1968) Calculating the acoustical room response by the use of a ray tracing technique. *Journal of Sound and Vibration* 8(1): 118–125.

Lopez JJ, Carnicero D, Ferrando N, et al. (2013) Parallelization of the finite-difference time-domain method for room acoustics modelling based on CUDA. *Mathematical and Computer Modelling* 57(7): 1822–1831.

Medina DS, St-Cyr A and Warburton T (2014) *OCCA: A Unified Approach to Multi-Threading Languages*. arXiv: 1403.0968 [cs.DC].

Miki Y (1990) Acoustical properties of porous materials—modifications of Delany-Bazley models, *Journal of the Acoustical Society of Japan* vol. 11(1): 19–24.

Morales N, Mehra R and Manocha D (2015) A parallel time-domain wave simulator based on rectangular decomposition for distributed memory architectures. *Applied Acoustics* 97(10): 104–114.

Pind F, Engsig-Karup AP, Jeong C-H, et al. (2019) Time domain room acoustic simulations using the spectral element method. *Journal of the Acoustical Society of America* 145(6): 3299–3310.

Saarelma J, Botts J, Hamilton B, et al. (2016) Audibility of dispersion error in room acoustic finite-difference time-domain simulation as a function of simulation distance, *Journal of the Acoustical Society of America* 139(4): pp. 1822–1832.

Sakamoto S (2007) Phase-error analysis of high-order finite difference time domain scheme and its influence on calculation results of impulse response in closed sound field. *Acoustical Science and Technology* 28(5): 295–309.

Sampedro Llopis H, Engsig-Karup AP, Jeong C-H, et al. (2022) Reduced basis methods for numerical room acoustic simulations with parametrized boundaries. *Journal of the Acoustical Society of America* 152: 851–865.

Savioja L and Svensson UP (2015) Overview of geometrical room acoustic modeling techniques. *Journal of the Acoustical Society of America* 138(2): 708–730.

Schoeder S, Wall WA and Kronbichler M (2019) ExWave: A high performance discontinuous Galerkin solver for the acoustic wave equation. *SoftwareX* 9: 49–54.

Spa C, Rey A and Hernandez E (2015) A GPU implementation of an explicit compact FDTD algorithm with a digital impedance filter for room acoustics applications. *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 23(8): 1368–1380.

Świrydowicz K, Chalmers N, Karakus A, et al. (2019) Acceleration of tensor-product operations for high-order finite

element methods. *The International Journal of High Performance Computing Applications* 33(4): 725–757.

Takahashi T and Hamada T (2009) GPU-accelerated boundary element method for Helmholtz' equation in three dimensions, *International Journal for Numerical Methods in Engineering* 80(10): 1295–1321.

Thomasson S-I (1976) Reflection of waves from a point source by an impedance boundary. *Journal of the Acoustical Society of America* 59(4): 780–785.

Wang H, Sihar I, Pagan Munoz R, et al. (2019) Room acoustics modelling in the time-domain with the nodal discontinuous Galerkin method, *Journal of the Acoustical Society of America* 145(4): 2650–2663.

Wang H, Yang J and Hornikx M (2020) Frequency-dependent transmission boundary condition in the acoustic time-domain nodal discontinuous Galerkin model. *Applied Acoustics* 164: 107280.

Warburton T (2006) An explicit construction of interpolation nodes on the simplex. *Journal of Engineering Mathematics* 56(3): 247–262.

Author biographies

Anders Melanders is a PhD student at the Section of Scientific Computing, Department of Applied Mathematics and Computer Science, Technical University of Denmark. His research interests include high-order spectral methods applied to acoustics and hydrodynamics with a focus on free surface flows and fluid dynamics, along with high-performance computing.

Emil Strøm got his master degree in engineering from the Technical University of Denmark. He currently works for Dalux, a software company in the construction industry. His work includes creating high-performing software for processing large amounts of user captured 360-videos.

Finnur Pind is the founder/CEO of Treble Technologies. Treble develops sound simulation technology that is revolutionizing how several sectors design and work with sound: architecture, consumer electronics, voice technology, metaverse and automotive. Finnur has a PhD in acoustics engineering from the Technical University of Denmark. Prior to founding Treble, he worked as a software engineer in telecom and as an acoustics consultant in the building industry, including spending four years at the prestigious architecture firm Henning Larsen.

Allan Peter Engsig-Karup is a Professor (Associate) of Computational Mathematics at the Section of Scientific Computing, Department of Applied Mathematics and Computer Science, Technical University of Denmark. He is a faculty member of The Danish Center for Applied Mathematics and Mechanics (DCAMM) and a faculty member of Center For Energy Resources Engineering (CERE). His research interests include high-performance computing using multi-device GPU acceleration, numerical analysis, efficient and scalable algorithms, development of new efficient simulation tools based on high order numerical methods and with application within engineering areas related to computational acoustics, computational hydrodynamics and in particular free surface flows, fluid dynamics, and energy resource areas (fx. renewables and reservoir simulation).

Cheol Ho Jeong is associate professor at the department of electrical and photonics engineering, Technical University of Denmark. He research area spans computational room acoustics, Reduced order modeling, inverse characterization of boundary conditions in acoustics, virtual prototyping and data-driven methods.

Tim Warburton is a Professor of Mathematics, Professor of Computational Modeling and Data Analytics, and holds the John K. Costain Faculty Chair in the College of Science at Virginia Tech. His research interests include high-performance computing, GPU acceleration, performance portability, and high-order finite element methods applied to computational modeling of acoustics, electromagnetics, fluid dynamics, and plasma physics.

Noel Chalmers received his PhD in Applied Mathematics from the University of Waterloo. He currently works at Advanced Micro Devices (AMD) Research, developing high-performance computing scientific software for exascale computing. His research centers on GPU-acceleration of scientific computing applications, high-order finite element methods, and scalable multilevel preconditioning algorithms.

Jan S Hesthaven after receiving his PhD in 1995 from the Technical University of Denmark, Professor Hesthaven joined Brown University, USA where he became Professor of Applied Mathematics in 2005. Since 2013 he has been Professor of Mathematics at EPFL, CH and holds the Chair of Computational Mathematics and Simulation Science. Since 2021 he has served as Provost of EPFL. His research interests are in the development, analysis, and application of high-order accurate methods for the solution of complex time-dependent problems. He has published more than 175 journal papers and 4 books and is a Fellow of SIAM, AMS and the Royal Danish Academy of Sciences and Letters.

Appendix

A Numerical flux derivation

To derive the upwinding flux, a Riemann problem at each element edge must be solved. We start by rewriting (1) in the quasi-linear form of a hyperbolic system

$$\frac{\partial \mathbf{q}}{\partial t} + \nabla \cdot \mathbf{F}(\mathbf{q}) = \frac{\partial \mathbf{q}}{\partial t} + \mathbf{A}_j \frac{\partial \mathbf{q}}{\partial x_j} = 0 \tag{32}$$

where $\mathbf{q}(\mathbf{x}, t) = [u, v, w, p]^T$ is the acoustic variable vector, $j \in [x, y, z]$ is the Cartesian coordinate index and $\mathbf{A}_j$ is a constant flux Jacobian matrix, given as

$$\mathbf{A}_j = \begin{bmatrix} 0 & 0 & 0 & \frac{\delta_{xj}}{\rho} \\ 0 & 0 & 0 & \frac{\delta_{yj}}{\rho} \\ 0 & 0 & 0 & \frac{\delta_{zj}}{\rho} \\ \rho c^2 \delta_{xj} & \rho c^2 \delta_{yj} & \rho c^2 \delta_{zj} & 0 \end{bmatrix} \tag{33}$$

where δ_{ij} denotes the Kronecker delta function. The flux of the system is defined as

$$\mathbf{F} = \left[\mathbf{A}_x \mathbf{q}, \mathbf{A}_y \mathbf{q}, \mathbf{A}_z \mathbf{q}\right] \tag{34}$$

We are interested in the numerical flux along the outward pointing element boundary normal $\widehat{\mathbf{n}}$, which leads to the creation of the following operator

$$\mathbf{N} = \widehat{\mathbf{n}} \cdot \mathbf{F} = \widehat{n}_x \mathbf{A}_x + \widehat{n}_y \mathbf{A}_y + \widehat{n}_z \mathbf{A}_z \tag{35}$$

Applying an eigendecomposition on $\mathbf{N}$ yields

$$\mathbf{N} = \mathbf{R}\mathbf{D}\mathbf{R}^{-1} \tag{36}$$

where

$$\mathbf{R} = \begin{bmatrix} \frac{\widehat{n}_x}{\rho c} & -\frac{\widehat{n}_x}{\rho c} & -\frac{\widehat{n}_z}{\widehat{n}_x} & -\frac{\widehat{n}_y}{\widehat{n}_x} \\ \frac{\widehat{n}_y}{\rho c} & -\frac{\widehat{n}_y}{\rho c} & 0 & 1 \\ \frac{\widehat{n}_z}{\rho c} & -\frac{\widehat{n}_z}{\rho c} & 1 & 0 \\ 1 & 1 & 0 & 0 \end{bmatrix} \tag{36a}$$

$$\begin{aligned} \mathbf{D} &= \mathbf{D}^+ + \mathbf{D}^- \\ &= \begin{bmatrix} c & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & -c & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \end{aligned} \tag{36b}$$

Since all eigenvalues are real, the system is hyperbolic, and therefore the eigenvalues represent the speeds of propagation of information of the characteristic variables $\mathbf{b}$, defined as $\mathbf{R}^{-1}\mathbf{q} = \mathbf{b}$ (Eleuterio, 1999).

The properties of $\mathbf{D}^+$ and $\mathbf{D}^-$ results in the numerical upwind flux

$$(\widehat{\mathbf{n}} \cdot \mathbf{F})^* = \mathbf{R}\mathbf{D}^+\mathbf{R}^{-1}\mathbf{q}_h^- + \mathbf{R}\mathbf{D}^-\mathbf{R}^{-1}\mathbf{q}_h^+ \tag{37}$$

B Hardware and compilers

All numerical experiments are carried out using the HPC system at the Technical University of Denmark (DTU) Computing Center. Three computing nodes are used, where each node consists of:

- CPU: 2 × Intel Xeon Gold 6142 @ 2.60 GHz,
- GPU: 4 × NVIDIA Tesla V100 SXM2 32 GB,

With the three nodes being connected by InfiniBand. The compilers used for compilation are:

- GCC v7.4.0,
- OpenMPI v3.1.3,
- OCCA v1.0,
- CUDA compilation tools v10.0.130.

For the scalability tests, we will use the setups of 1, 2, 4, 8, and 12 GPUs. Note that when running tests with four or less GPUs, they will all be on the same node. This means that when going from 4 to 8 GPUs there will be an additional cost of transfer, as internal communication is not as fast as a memory copy within a node.

C Strong scaling measurements

C.1 Strong scaling runtime for N = 4.

Table 13. Runtime per time step in seconds for meshes in Table 3 with $N = 4$. Note that mesh 6 is not included, as memory requirements are too much for a single GPU.

#GPUs	Mesh 1	Mesh 2	Mesh 3	Mesh 4	Mesh 5
1	0.0076	0.0310	0.0920	0.1536	0.4618
2	0.0042	0.0158	0.0470	0.0778	0.2322
4	0.0043	0.0092	0.0237	0.0393	0.1163
8	0.0035	0.0080	0.0122	0.0200	0.0585
12	0.0035	0.0082	0.0097	0.0136	0.0393

C.2 Strong scaling runtime for N = 6.

Table 14. Runtime per time step in seconds for meshes 1 through 4 in Table 3 with $N = 6$.

#GPUs	Mesh 1	Mesh 2	Mesh 3	Mesh 4
1	0.0227	0.0960	0.2894	0.4803
2	0.0119	0.0485	0.1447	0.2429
4	0.0083	0.0247	0.0728	0.1208
8	0.0066	0.0160	0.0371	0.0605
12	0.0062	0.0164	0.0249	0.0408