Massively Parallel Nodal Discontinuous Galerkin Finite Element Method Simulator for Room Acoustics

Anders Melander¹, Emil Strøm¹, Finnur Pind²,³, Allan P. Engsig-Karup¹, Cheol-Ho Jeong³, Tim Warburton⁴, Noel Chalmers⁴ and Jan S. Hesthaven⁵

Abstract
We present a massively parallel and scalable nodal Discontinuous Galerkin Finite Element Method (DGFEM) solver for the time-domain linearised 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 using massively parallel computing.

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, e.g., 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 m³ to 30 000 m³, 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) to 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 (Krookstad 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.

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, 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).

¹Scientific Computing Section, Department of Applied Mathematics and Computer Science, Technical University of Denmark, Kgs. Lyngby, Denmark.
²Henning Larsen, Copenhagen, Denmark.
³Acoustic Technology Group, Department of Electrical Engineering, Technical University of Denmark, Kgs. Lyngby, Denmark.
⁴Department of Mathematics, Virginia Tech, United States.
⁵Chair of Computational Mathematics and Simulation Science, cole Polytechnique Fdrale de Lausanne, Lausanne, Switzerland.

Corresponding author: Anders Melander
Email: anders.d.melander@gmail.com
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). 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, 1,000 m³) 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. There is some past work on the HPC implementation of wave-based room acoustic simulations. 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. 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 (BCs).

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

\[ \mathbf{v}_t = -\frac{1}{\rho} \nabla p, \]

\[ p_{,t} = -\rho c^2 \nabla \cdot \mathbf{v}, \]

where \( \mathbf{v}(x, t) \) [m/s] is the particle velocity at position \( x \) in the domain \( \Omega \) at time \( t \), \( p(x, t) \) is the acoustic pressure [Pa], \( \rho \) is the density of air and \( c \) is the speed of sound in air. In this work, we use \( \rho = 1.2 \text{kg/m}^3 \) and \( c = 343 \text{ m/s} \). We let \( u, v \) and \( w \) be the Cartesian coordinates \( x, y, z \) 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 \( \partial \Omega \). In the following, we will introduce the boundary condition types that have been implemented in the DGFEM solver. These are rigid, frequency independent and frequency dependent boundary conditions.
2.1.1 Rigid The rigid boundary condition perfectly reflects the impinging wave back into the domain \( \Omega \). Mathematically this is expressed as
\[
\mathbf{n} \cdot \mathbf{v} = 0 \quad \text{on} \quad \partial \Omega, \tag{2}
\]
where \( \mathbf{n} \) is the outward pointing normal of the boundary surface \( \partial \Omega \).

2.1.2 Locally-reacting frequency independent The frequency independent impedance boundary condition adds absorption to the domain boundaries, with all frequencies being absorbed equally. The absorption is dictated by the normal incidence surface impedance \( Z \). The normal velocity at the boundary becomes
\[
\mathbf{n} \cdot \mathbf{v} = v_n = \frac{p}{Z} = pY, \tag{3}
\]
where \( Y \) is the normal boundary admittance. Using high values of \( Z \) implies a highly reflective boundary conditions, whereas values close to the characteristic impedance of the air \( Z_{\text{air}} = \rho c \) results in a highly absorptive boundary.

2.1.3 Locally-reacting frequency dependent Most real world surfaces exhibit frequency dependent absorption. Mathematically, this can be written as
\[
\hat{v}_n = \frac{\hat{p}(\omega)}{Z(\omega)} = \hat{p}(\omega)Y(\omega), \tag{4}
\]
where \( \omega \) is the angular frequency and \( (\hat{v}_n, \hat{p}) \) denote the Fourier transformed normal velocity and pressure, respectively. \( Z(\omega) \) is no longer constant, as in the frequency independent case, but is a complex function of the angular frequency. For a particular surface, \( Z(\omega) \) can be measured, e.g., in an impedance tube or using pressure-velocity sensors. Alternatively, several material models exist to estimate surface impedance of many common materials and constructions.

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 (Pind et al. 2019; Dragna et al. 2015). 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
\[
Y(\omega) = Y_\infty + \sum_{k=1}^{Q} \frac{A_k}{\alpha_k - j\beta_k - j\omega} + \sum_{k=1}^{S} \left( \frac{B_k + jC_k}{\alpha_k + j\beta_k - j\omega} + \frac{B_k - jC_k}{\alpha_k - j\beta_k - j\omega} \right). \tag{6}
\]
Here \( Q \) is the number of real poles, \( S \) is the number of complex conjugate pole pairs of \( Y(\omega) \), \( \lambda_k \) are the real poles, \( \alpha_k \pm j\beta_k \) are the complex conjugate pole pairs, and \( Y_\infty \), \( A_k \), \( B_k \) and \( C_k \) are numerical coefficients. For causality, \( \lambda_k \) and \( \alpha_k \) 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 (4), 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
\[
v_n(t) = Y_\infty p(t) + \sum_{k=1}^{Q} A_k \phi_k(t) + \sum_{k=1}^{S} 2 \left( B_k \psi_k^{(1)}(t) + C_k \psi_k^{(2)}(t) \right), \tag{8}
\]
where \( \phi_k \) 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)
\[
\begin{align*}
\frac{d\phi_k}{dt} &\lambda_k \phi_k = p(t), \\
\frac{d\psi_k^{(1)}}{dt} &\alpha_k \psi_k^{(1)} + \beta_k \psi_k^{(2)} = p(t), \\
\frac{d\psi_k^{(2)}}{dt} &\alpha_k \psi_k^{(2)} - \beta_k \psi_k^{(1)} = 0.
\end{align*} \tag{9}
\]
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 \( N_{\text{poles}} \) 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. DGFEM is a high-order method (Hesthaven and Warburton 2008) capable of solving the time-domain acoustic partial differential equations within a domain \( \Omega \), subject to an appropriate set of boundary and initial conditions. The spatial domain is tessellated by \( K \) non-overlapping elements as

\[
\Omega \simeq \Omega_h = \bigcup_{k=1}^K D^k.
\]  

(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 \( v \) and \( p \) are expressed on the \( k \)'th element in terms of polynomial basis functions and corresponding modal or nodal coefficients. This is expressed as

\[
v^k_h(x, t) = \sum_{n=1}^{N_v} \tilde{v}_n^k(t) \phi_n(x) = \sum_{n=1}^{N_v} \tilde{v}_n^k(x^h_n, t) \ell^k_n(x),
\]

\[
p^k_h(x, t) = \sum_{n=1}^{N_p} \tilde{p}_n^k(t) \phi_n(x) = \sum_{n=1}^{N_p} \tilde{p}_n^k(x^h_n, t) \ell^k_n(x).
\]  

(11)

Here \( \phi_n(x) \) is the hierarchical modal basis function and \( \ell^k_n(x) \) is the corresponding nodal Lagrange polynomial basis function on element \( D^k \). \( \tilde{v}_n^k(t) \) and \( \tilde{p}_n^k(t) \) are the modal coefficients for the modal basis functions, and \( \tilde{v}_n^k(x^h_n, t) \) and \( \tilde{p}_n^k(x^h_n, t) \) are the corresponding nodal coefficients for the nodal basis functions. The modal basis functions are assumed to be orthonormal as this simplifies several needed computations. On each element \( x^h_k \) 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 \( \alpha \)-optimized nodal distribution is used due to its low Lebesque constants (Hesthaven and Teng 2000). The solutions on the elements are used to approximate the global solutions \( v \) and \( p \) in terms of direct sums

\[
v \approx v_h = \bigoplus_{k=1}^K v^k_h, \quad p \approx p_h = \bigoplus_{k=1}^K p^k_h.
\]  

(12)

The strong formulation of the governing equations, when using the nodal basis series to represent the unknown variables \( v^k_h \) and \( p^k_h \), takes the following form

\[
\int_{D^k} \frac{\partial v^k_h}{\partial t} \ell^k_n(x) dx = -\frac{1}{\rho} \int_{D^k} \nabla \cdot \nabla \cdot \tilde{v}_n^k(x) dx + \frac{1}{\rho} \int_{\partial D^k} \tilde{n} \cdot (p^k_h - p^*) \ell^k_n(x) dx,
\]

\[
\int_{D^k} \frac{\partial p^k_h}{\partial t} \ell^k_n(x) dx = -\int_{D^k} \rho \mathbf{c}^2 \nabla \cdot \tilde{v}_n^k(x) dx + \int_{\partial D^k} \tilde{n} \cdot (\tilde{v}_n^k - \tilde{v}^*) \ell^k_n(x) dx.
\]  

(13)

where the star-labeled terms denote numerical fluxes that propagate the solution between elements. This can be rewritten in a discrete formulation using matrix-based operators for integration and differentiation

\[
\frac{du^k}{dt} = -\frac{1}{\rho} D_x^k p^k_h + \frac{1}{\rho} \mathcal{L}_x (p^k_h - p^*) \hat{n}_x,
\]

\[
\frac{dv^k}{dt} = -\frac{1}{\rho} D_y^k p^k_h + \frac{1}{\rho} \mathcal{L}_y (p^k_h - p^*) \hat{n}_y,
\]

\[
\frac{dp^k}{dt} = -\frac{1}{\rho} D_z^k p^k_h + \frac{1}{\rho} \mathcal{L}_z (p^k_h - p^*) \hat{n}_z,
\]

\[
\frac{dp^k}{dt} = -\rho \mathbf{c}^2 (D_x \tilde{u}^k + D_y \tilde{v}^k + D_z \tilde{w}^k) + \rho \mathbf{c}^2 \mathcal{L}^k \hat{n} \cdot (\tilde{v}^k - \tilde{v}^*).
\]  

(14)

The matrix operations on the right hand side are expressed in terms of the operators defined on the reference elements

\[
D_x = \mathcal{V}_x \mathcal{V}^{-1}, \quad D_y = \mathcal{V}_y \mathcal{V}^{-1}, \quad D_z = \mathcal{V}_t \mathcal{V}^{-1}, \quad \mathcal{M}(\mathcal{V} \mathcal{V}^T)^{-1}.
\]  

(15)

Here \( \mathcal{V}_{ij} \equiv \phi_j(x_i) \) is the generalized Vandermonde matrix and \( (\mathcal{V} \mathcal{X}) \) \( ij \equiv \partial \phi_j(x_i)/\partial X \) are defined in terms of reference element coordinates \( X = r, s, t \).

These operators can, via the chain rule of the continuous variables, e.g.,

\[
\frac{\partial f(x)}{\partial x} = \frac{\partial x}{\partial \xi} \frac{\partial f(x)}{\partial \xi} + \frac{\partial s}{\partial \xi} \frac{\partial f(x)}{\partial s} + \frac{\partial t}{\partial \xi} \frac{\partial f(x)}{\partial t}.
\]  

(16)

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

\[
D_x^k = r_x^k D_x + s_x^k D_y + t_x^k D_z,
\]

\[
D_y^k = r_y^k D_x + s_y^k D_y + t_y^k D_z,
\]

\[
D_z^k = r_z^k D_x + s_z^k D_y + t_z^k D_z.
\]  

(17)

Furthermore, we introduce the lift operator \( \mathcal{L} \) needed for the incorporation of boundary conditions via numerical fluxes at
the element edges
\[ \mathcal{L} = (\mathcal{M})^{-1} \mathcal{E}, \quad \mathcal{M} = (\nabla \mathcal{V})^{-1}, \quad (18) \]
where \( \mathcal{E} \) is the column-wise concatenation of mass matrices defined in terms of the nodal distributions on the faces.

To avoid computing and storing these matrices for every element, the matrix operators are determined for a reference element, and a geometric mapping from the arbitrary mesh elements to the reference elements is used, e.g.,
\[ \mathcal{M}^k = J^k (\nabla \mathcal{V})^{-1}, \quad (19) \]
where \( J^k \) is a diagonal matrix storing the computed Jacobian values of the local map of the \( k \)'th element to the reference element. Hence, we need to compute and store the \( J^k \)'s for each element, and the procedure in this way treats curvilinear and straight-sided elements in a similar way. By utilizing matrix-based integration via the mass matrices in the numerical scheme all integrals in the weak formulation is exact to twice the polynomial order of the polynomial basis. For curvilinear elements where the Jacobians are non-constant, this may lead to cubature errors in the operators.

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. We start by rewriting (1) in the quasi-linear form of a hyperbolic system
\[ \frac{\partial q}{\partial t} + \nabla \cdot \mathbf{F}(q) = \frac{\partial q}{\partial t} + \mathbf{A}_j \frac{\partial q}{\partial x_j} = 0, \quad (20) \]
where \( q(x, t) = [u, v, w, p]^T \) 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
\[ \mathbf{A}_j = \begin{bmatrix} 0 & 0 & 0 & \frac{\partial x_j}{\partial x} \\ 0 & 0 & 0 & \frac{\partial x_j}{\partial y} \\ \frac{\partial x_j}{\partial y} & 0 & 0 & 0 \\ \frac{\partial x_j}{\partial z} & 0 & 0 & 0 \end{bmatrix}, \quad (21) \]
where \( \delta_{ij} \) denotes the Kronecker delta function. The flux of the system is defined as
\[ \mathbf{F} = [\mathbf{A}_x q, \mathbf{A}_y q, \mathbf{A}_z q]. \quad (22) \]
We are interested in the numerical flux along the outward pointing element boundary normal \( \mathbf{n} \), which leads to the creation of the following operator
\[ \mathbf{N} = \mathbf{n} \cdot \mathbf{F} = n_x \mathbf{A}_x + n_y \mathbf{A}_y + n_z \mathbf{A}_z. \quad (23) \]
Applying an eigendecomposition on \( \mathbf{N} \) yields
\[ \mathbf{N} = \mathbf{R} \mathbf{D} \mathbf{R}^{-1}, \quad (24) \]
where
\[ \mathbf{R} = \begin{bmatrix} \frac{\partial x}{\partial n} & -\frac{\partial x}{\partial n} & -\frac{\partial x}{\partial n} & -\frac{\partial x}{\partial n} \\ \frac{\partial x}{\partial n} & \frac{\partial x}{\partial n} & 0 & 1 \\ \frac{\partial x}{\partial n} & \frac{\partial x}{\partial n} & 1 & 0 \\ 1 & 1 & 0 & 0 \end{bmatrix}, \quad (25) \]
\[ \mathbf{D} = \begin{bmatrix} c & 0 & 0 & 0 \\ 0 & -c & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}. \]
Here the diagonal of \( \mathbf{D} \) contains the eigenvalues of \( \mathbf{N} \). 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).

By splitting the eigenvalue matrix \( \mathbf{D} \) into its positive and negative parts
\[ \mathbf{D} = \mathbf{D}^+ + \mathbf{D}^- \]
\[ = \begin{bmatrix} c & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & -c & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \quad (26) \]
\( \mathbf{D}^+ \) corresponds to the parts of \( \mathbf{b} \) where the direction is along the normal vector \( \mathbf{n} \), meaning the information is leaving the element, whereas \( \mathbf{D}^- \) corresponds to the parts of \( \mathbf{b} \) where the direction is opposite \( \mathbf{n} \), implying that the information is entering the element. In the numerical flux we seek to combine the information from the current element boundary point, given in \( \mathbf{q}_h \), with the information from the neighbouring element boundary point, given in \( \mathbf{q}_h \). Since \( \mathbf{n} \cdot \mathbf{F} = \mathbf{N} \mathbf{q}_h \), the following holds
\[ \mathbf{n} \cdot \mathbf{F} = \mathbf{R} \mathbf{D}^{-1} \mathbf{q}_h = \mathbf{R}^+ \mathbf{R}^{-1} \mathbf{q}_h + \mathbf{R}^- \mathbf{R}^{-1} \mathbf{q}_h. \]
By the properties of $D^+$ and $D^-$, this results in the numerical upwind flux
\[
(\hat{\mathbf{n}} \cdot \mathbf{F})^* = \mathbf{R}D^+ \mathbf{R}^{-1} \mathbf{q}_h^- + \mathbf{R}D^- \mathbf{R}^{-1} \mathbf{q}_h^+.
\] (28)

Boundary conditions are weakly enforced through the numerical flux terms. For element faces that lie on the domain boundary, $\mathbf{q}_h^+$ needs to be defined according to the boundary conditions. For surface impedance boundary conditions, an averaging approach is used to impose Dirichlet conditions, where the neighboring value is given by $p_h^* = p_h^-$ and $v_h^* = -v_h^- + 2g_v$, with $v_h^-$ given by either (3) or (8). Thus, at the domain boundary nodes, the flux term in (14d) becomes $\hat{\mathbf{n}} \cdot (\mathbf{v}_h^* - \mathbf{v}^*) = \hat{\mathbf{n}} \cdot \mathbf{v}_h^* - \mathbf{v}_h^-$.

3.2 Temporal discretization and stability

The semi-discrete system in (14) can be expressed as a general form ODE
\[
\frac{d \mathbf{q}_h}{dt} = \mathcal{L}(\mathbf{q}_h(t), t),
\] (29)

where $\mathcal{L}$ is the spatial operator. This ODE system is integrated in time using a low-storage 4th order explicit Runge-Kutta time-stepping method,
\[
\begin{aligned}
\mathbf{q}_h^{(0)} &= \mathbf{q}_h^n, \\
i &\in [1, \ldots, 5] : \\
\mathbf{q}_h^{(i)} &= \mathbf{q}_h^{(i-1)} + \Delta t \mathcal{L}(\mathbf{q}_h^{(i-1)}, t^n + c_i \Delta t), \\
\mathbf{q}_h^{n+1} &= \mathbf{q}_h^{(5)},
\end{aligned}
\] (30)

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 $\lambda_e$ are the eigenvalues 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 spectrum of the matrix operators, and 2) the stiffness of the ADEs. In DGFEM, the matrix operator eigenvalue spectrum scales as $\max |\lambda_e| \sim C_2 \gamma^N$, where the $C_2$ constant is dictated by the size of the smallest mesh element and $\gamma$ 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
\[
\Delta t = C_{\text{CHL}} \min(\Delta x_i) \frac{1}{c \gamma N^2},
\] (31)

where $\Delta x_i$ is the smallest edge length of the mesh elements and $C_{\text{CHL}}$ is a constant of $O(1)$.

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 $\Omega_h$ 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 simulator then adds the $\alpha$-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.

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 $\alpha$-optimized curvilinear node positions. This can be 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 $\alpha$-optimized and equidistant node distributions, respectively. Using a hierarchical modal basis, the coefficients $\mathbf{r}$ of this representation can be determined from
\[
\mathbf{r} = \mathcal{V} \hat{\mathbf{r}}, \quad \mathbf{r}^e = \mathcal{V}^e \hat{\mathbf{r}},
\] (32)
where \((V^e)_{ij} = \phi_j(r^e_i)\), using same basis used to define \(V\) but defined in terms of the different set of equidistant nodes. This implies the relationship between the node distributions

\[
\mathbf{r} = V^{-1}(V^e)\mathbf{r}^e
\]  

from which an interpolation matrix \(I : \mathbf{r}^e \rightarrow \mathbf{r}\) is defined as

\[
I = V^{-1}(V^e)^{-1}.
\]

Using this operator offers a way to go from the \texttt{gmsh} generated curvilinear mesh with equidistant node distribution to a curvilinear mesh with \(\alpha\)-optimized node distribution.

4 HPC implementation

The proposed HPC room acoustics simulator is based on the open source \texttt{libParanumal} framework (Karakus et al. 2019; Wirydowicz 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. In practice this means that \texttt{libParanumal} splits the domain into different chunks. This is illustrated in Fig. 1. Each CPU core only stores the simulation information needed for the computations in elements belonging to its own part of the domain. However, as Fig. 1 illustrates, the CPUs share an interface between them, where information has to be shared for the acoustic wave to propagate from one CPU to the next. This means that for some elements, the numerical flux has to be computed based on neighbouring elements that are located on a different CPU, requiring that MPI communication between cores must take place. This is called a halo exchange. The same communication is used when GPUs are used.

When executing the code on GPUs, the code is structured such that each CPU core is assumed to have access to one GPU, e.g., 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 2, 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 (30), including all the interface communication, is summarized in Algorithm 1. The GPU functions, called 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 $q_h^{i-1}$

Result: An updated stage $q_h^{i}$

1. GPU to CPU memory copy of interface nodes (CPU/GPU). Simultaneously initiate volume integral compute kernel (GPU).
2. Initiate the interface MPI communication (CPU).
3. CPU to GPU memory copy of interface nodes (CPU/GPU).
4. Initiate 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 of the right hand side of (14) can start simultaneously, as shown in step 1. Ideally, the communication work in steps 1, 2 and 3 is finished before the volume kernel finishes. This way, communication costs are completely hidden. This is possible because the volume kernel is 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. Overall, the amount of communication required in each time-step is considerable for high resolution simulations, which means that it cannot be expected that the simulator demonstrates perfect scaling on multiple GPUs.

### 5 Numerical experiments

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 4 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.

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.

#### 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

$$
p(x, 0) = \exp\left(-\frac{(x-x_0)^2 + (y-y_0)^2 + (z-z_0)^2}{s_{xyz}}\right)
$$

where $(x_0, y_0, z_0)$ is the source position and $s_{xyz} = 0.3$ 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, i.e., the error behavior as a function of the basis function order for a fixed mesh, and $h$-convergence, i.e., the error behavior as a function of the mesh element size. The numerical error is defined as $e = \|p^{DGFEM} - p^{TS}\|_\infty$, 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 $C_{ CFL} = 1$ in (31).

Figure 3 shows the resulting $p$-convergence, tested for three different meshes, having varying resolution. The figure reveals the expected spectral convergence, i.e., an exponentially fast decay of errors, is indeed observed.
Figure 4. $h$-convergence for the rigid cube test case.

Figure 4 shows the $h$-convergence for different orders of the basis. Table 1 lists the resulting convergence rates. The table also includes convergence rates for a case where the time step size has been reduced significantly ($C_{CFL} = 0.02$). 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.

<table>
<thead>
<tr>
<th>$N$</th>
<th>$C_{CFL} = 1$</th>
<th>$C_{CFL} = 0.02$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-0.778</td>
<td>-0.778</td>
</tr>
<tr>
<td>2</td>
<td>-3.239</td>
<td>-3.240</td>
</tr>
<tr>
<td>3</td>
<td>-3.919</td>
<td>-3.920</td>
</tr>
<tr>
<td>4</td>
<td>-4.618</td>
<td>-4.620</td>
</tr>
<tr>
<td>5</td>
<td>-5.752</td>
<td>-5.752</td>
</tr>
<tr>
<td>6</td>
<td>-6.188</td>
<td>-6.188</td>
</tr>
<tr>
<td>7</td>
<td>-7.754</td>
<td>-7.747</td>
</tr>
<tr>
<td>8</td>
<td>-8.161</td>
<td>-8.166</td>
</tr>
</tbody>
</table>

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

5.1.2 Cylinder domain Consider now a cylinder shaped domain, of 1 m height and 1 m radius and having rigid boundaries. A convergence test is carried out, where the domain is 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) and $N = 8$ basis function order. The numerical error is then defined as $e = \| p^{DGFEM} - p^{DGFEM}_{\text{fine}} \|$, where $p^{DGFEM}$ is the simulated solution in 10 randomly chosen receiver points at $t = 0.01$ and $p^{DGFEM}_{\text{fine}}$ is the simulated solution in the same receiver points for the fine mesh reference simulation.

Figure 5 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 6, 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. 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). However, in practical room acoustics simulations, error levels below $10^{-7}$ are rarely needed and, furthermore, such anti-aliasing techniques increases the computational cost of the scheme.

Figure 6. $h$ convergence for the cylinder geometry using either affine or curvilinear meshes.

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 $\times$ 10 m $\times$ 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 $t_{\text{final}} = 0.02$ 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 $s_{x,y,z} = 0.2$ 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 are estimated using Miki’s model and a transfer matrix method (Miki 1990; Allard and Atalla 2009). 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

$$\epsilon = \frac{\mid Y(f) - Y_{\text{in}}(f) \mid}{Y(f)},$$

where $Y$ 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.

<table>
<thead>
<tr>
<th>ID</th>
<th>$d_{\text{mat}}$</th>
<th>$d_0$</th>
<th>$\sigma_{\text{mat}}$</th>
<th>$N_{\text{poles}}$</th>
<th>$\epsilon_{\text{Re}}$</th>
<th>$\epsilon_{\text{Im}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>BC1</td>
<td>0.02</td>
<td>0.2</td>
<td>50 000</td>
<td>8</td>
<td>0.0216</td>
<td>0.0177</td>
</tr>
<tr>
<td>BC2</td>
<td>0.05</td>
<td>0.0</td>
<td>10 000</td>
<td>4</td>
<td>0.0213</td>
<td>0.0071</td>
</tr>
</tbody>
</table>

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

Figure 7 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.

5.3 Heterogeneous multi-device CPU-GPU performance

5.3.1 Strong scaling Strong scaling is characterised 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.

Four meshes of varying resolutions are constructed, described in Table 3. These meshes are used to analyse 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

$$\eta_S = \frac{T_R^B M^B}{T_R M},$$

(37)

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 $M$. In practice, perfect scaling is impossible, as there is always some latency introduced when scaling up the number of processors used.

An array of simulations is carried out, using the meshes defined in Table 3, and the runtime is recorded. The resulting strong scaling computational efficiency is shown in Table 4.

<table>
<thead>
<tr>
<th>#GPUs</th>
<th>Mesh 1</th>
<th>Mesh 2</th>
<th>Mesh 3</th>
<th>Mesh 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>0.630</td>
<td>0.903</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>0.393</td>
<td>0.656</td>
<td>0.851</td>
<td>0.913</td>
</tr>
<tr>
<td>8</td>
<td>0.290</td>
<td>0.586</td>
<td>0.704</td>
<td>0.743</td>
</tr>
<tr>
<td>12</td>
<td>0.159</td>
<td>0.384</td>
<td>0.505</td>
<td>0.573</td>
</tr>
</tbody>
</table>

Table 4. Strong scaling computational efficiency $\eta_S$ for the meshes in Table 3 with $N = 4$. "-" denotes that running the simulation was not possible due to it requiring more memory than the GPUs in the setup have.

As two of the meshes with $N = 4$ were not able to be run on 1 GPU due to memory limitations, a comparison where using $M^B = 2$ GPUs as a baseline is shown in table 5. These numbers are more directly comparable between meshes. As expected, good scaling occurs for high GPU counts, provided a high resolution mesh is used to ensure that the efficiency does not suffer due to the overhead associated with communication and synchronization of the cores.

<table>
<thead>
<tr>
<th>#GPUs</th>
<th>Mesh 1</th>
<th>Mesh 2</th>
<th>Mesh 3</th>
<th>Mesh 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>0.623</td>
<td>0.727</td>
<td>0.851</td>
<td>0.913</td>
</tr>
<tr>
<td>8</td>
<td>0.460</td>
<td>0.649</td>
<td>0.704</td>
<td>0.743</td>
</tr>
<tr>
<td>12</td>
<td>0.252</td>
<td>0.425</td>
<td>0.505</td>
<td>0.574</td>
</tr>
</tbody>
</table>

Table 5. Strong scaling computational efficiency $\eta_S$ for the meshes in Table 3 with $N = 4$ and using $M^B = 2$ GPUs as the baseline.

In Figure 8, the speed-up factors with $M^B = 2$ GPUs as a baseline can be seen. Note that when going from 8 GPUs to 12 GPUs for mesh 1 and 2, the communication time begins to dominate the runtime. The GPUs do not have a high enough workload, resulting in them finishing the volume kernel computation of step 1 in Algorithm 1 before the CPUs have finished the communications in step 3.

Figure 8. Speed-up factors, with the 2 GPU configuration as a baseline.
involved. Because of the increased size of the problem, meshes 3 and 4 require a minimum of 4 and 8 GPUs, respectively. Thus, only the strong scaling performance for meshes 1 and 2 is considered. Table 6 shows the strong scaling efficiency factor $\eta_S$. The scaling efficiency has improved, as compared to the $N = 4$ case, due to the increase in DoF.

<table>
<thead>
<tr>
<th>#GPUs</th>
<th>Mesh 1</th>
<th>Mesh 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0.813</td>
<td>0.923</td>
</tr>
<tr>
<td>4</td>
<td>0.525</td>
<td>0.859</td>
</tr>
<tr>
<td>8</td>
<td>0.390</td>
<td>0.700</td>
</tr>
<tr>
<td>12</td>
<td>0.235</td>
<td>0.486</td>
</tr>
</tbody>
</table>

Table 6. Strong scaling computational efficiency $\eta_S$ for two of the meshes in Table 3 with $N = 6$.

We combine the information about mesh 1 and 2 from Table 4 and 6 in Figure 9. Here we clearly see the efficiency increase when comparing $N = 4$ to $N = 6$ for both of the two meshes.

5.3.2 Weak scaling

Weak scaling is characterised by how the runtime changes when fixing the problem size for each processing unit. Thus the problem size is increased when increasing the number of processing units, to keep the overall workload constant on each processing unit. The weak scaling computational efficiency is given by

$$\eta_W = \frac{T_B}{T_R}$$

(38)

The nature of weak scaling requires us to construct new meshes, these meshes are shown in Table 7. The meshes are structured meshes, with around 55 000 elements per GPU.

<table>
<thead>
<tr>
<th># of elements</th>
<th>DoF</th>
<th>DoF</th>
<th>DoF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mesh 1</td>
<td>55k</td>
<td>1 925k</td>
<td>4 620k</td>
</tr>
<tr>
<td>Mesh 2</td>
<td>111k</td>
<td>3 885k</td>
<td>9 324k</td>
</tr>
<tr>
<td>Mesh 3</td>
<td>219k</td>
<td>7 665k</td>
<td>18 396k</td>
</tr>
<tr>
<td>Mesh 4</td>
<td>439k</td>
<td>15 365k</td>
<td>36 876k</td>
</tr>
<tr>
<td>Mesh 5</td>
<td>661k</td>
<td>23 135k</td>
<td>55 524k</td>
</tr>
</tbody>
</table>

Table 7. Meshe and resulting DoF used for weak scaling analysis, where k denotes kilo, i.e. $\times 1000$.

The weak scaling efficiency results are shown in Figure 10. The scaling improves with increasing basis function order. In particular, good scaling performance is achieved for $N = 8$. This behavior again shows that the scaling is very dependent on the overall resolution of the mesh, and that the GPUs will be bottle-necked by the communication for low resolutions. Having a higher basis function order corresponds to each GPU having a higher workload, which means that the communication time is no longer dominating the runtime. Increasing the resolution further beyond 9 million DoF per GPU would steadily move the weak scaling efficiency closer to 1.
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 2 in Table 3 and use $N = 6$. These are cube shaped domains (all bounding surfaces are equally large). We set the BCs such that two surfaces are frequency dependent impedance boundaries, with the BCs defined in Table 2, and the remaining four surfaces have frequency independent absorption with $Z = 10 000 \text{ Ns/m}^4$. This is meant to represent a more realistic simulation scenario.

The strong scaling results are shown in Table 8 and the relative change in runtime when having non-rigid BCs, as compared to the rigid case, is shown in Table 9. The runtime increases by an average of only 1.3%. These results show that having realistic boundary conditions only has a marginal influence on the performance. Thus the scaling results shown in Sections 5.3.1 and 5.3.2, will also hold for cases with realistic BCs. The runtime is more affected for the smaller mesh – this is expected since the boundaries are a larger proportion of this mesh.

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. It is therefore of interest to analyze how the runtime of the simulator grows as the valid frequency range of the simulation is increased. The valid frequency range of the simulation depends on the spatio-temporal resolution – a high enough resolution must be chosen such that numerical errors are below a certain threshold. A relevant threshold in room acoustic simulations is the audibility threshold for dispersion errors, which is 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 (Pind et al. 2019; Kreiss and Oliger 1972). 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 main numerical dispersion error below the audibility threshold. The 1D advection equation is

$$u_t + cu_x = 0, \quad (39)$$

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 $f(s) = e^{\imath s}$, 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

$$u^0 = u^N e^{-\hat{\omega} N \Delta t}, \quad (40)$$

where $\hat{\omega}$ is the numerical frequency, which will differ from
the exact frequency $\omega$ due to the dispersion of the numerical
scheme. This non-linear equation can be solved numerically
for $\hat{\omega}$, 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 $c_d/c = \hat{\omega}/\omega$, where $c_d$ is the numerical wave speed.

Figure 11 shows an example dispersion relationship for the
$N = 2, 4, 6$ basis function orders. In this dispersion analysis,
the time step is kept very small, $\Delta t = 10^{-5}$ s and wave speed
c = 1 m/s, to ensure that the spatial errors dominate. Thus,
the analysis only takes into account the spatial discretization
errors. The dispersion relation is measured after $n = 2$ time
steps. In practice, the dispersion relation will depend on both
spatial and temporal errors, and will, furthermore, vary with
the simulation duration, where the benefits of using high-
order basis functions versus low-order will increase with
increasing number of time steps (Kreiss and Oliger 1972).
In 3D simulations, the dispersion will also be direction
dependent. Taken together, the dispersion properties are
very case dependent. It is thus emphasized that this is only
intended as a rough estimate of the PPW needed to maintain
the desirable error levels, when using different basis function
orders.

![Figure 11. 1D dispersion analysis results when using
$\Delta t = 0.0001$, $c = 1$ m/s and $n = 2$ time steps.](image)

<table>
<thead>
<tr>
<th>Points per wavelength</th>
<th>$N = 2$</th>
<th>$N = 4$</th>
<th>$N = 6$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_{PW}$</td>
<td>17.7</td>
<td>7.3</td>
<td>6.0</td>
</tr>
</tbody>
</table>

Table 10. Points-per-wavelength needed to ensure spatial
dispersion errors remain under 2%, based on the 1D dispersion
analysis of figure 11.

Table 10 lists the PPW needed to ensure that spatial
dispersion errors remain under 2% for the dispersion analysis
example case considered. 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 analyse how the runtime changes
as the frequency range of the simulation is extended,
when using different basis function orders. Two rooms are
considered, a $4 \times 2.7 \times 3$ m ‘bedroom’ and a $9 \times 7 \times 3$ 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 for the
two rooms. 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 $N = 2$, it was not
possible to go beyond $f_{max} = 1000$ Hz, because the runtime
was longer than 24 hours, which we set as the maximum
allowed runtime for the tests. A function on the form
$r = a f_{max}^n$ is fitted to the measured runtimes, where $r$ is the
estimated runtime. This is illustrated with the dashed lines
in the figure. The resulting fitted $a$ coefficients are shown
in Table 11. As can be seen, $a$ is considerably lower when
using the higher order basis functions. The $R^2$ measure of
the model fit quality is also presented, in all cases a fairly
good fit is observed.

![Figure 12. Measured runtime as a function of simulation
frequency range $f_{max}$. The dashed lines are the fitted runtime
model on the form $r = a f_{max}^n$.](image)
Table 11. Coefficients $a$ for the runtime model $r = a r_{\text{max}}$ and associated $R^2$ values for evaluating the goodness of the fit.

<table>
<thead>
<tr>
<th>$N$</th>
<th>$a$</th>
<th>$R^2$</th>
<th>$a$</th>
<th>$R^2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>$1.04 \cdot 10^{-8}$</td>
<td>0.92</td>
<td>$2.74 \cdot 10^{-8}$</td>
<td>0.96</td>
</tr>
<tr>
<td>4</td>
<td>$2.11 \cdot 10^{-10}$</td>
<td>0.86</td>
<td>$8.36 \cdot 10^{-10}$</td>
<td>0.91</td>
</tr>
<tr>
<td>6</td>
<td>$1.06 \cdot 10^{-10}$</td>
<td>0.85</td>
<td>$3.28 \cdot 10^{-10}$</td>
<td>0.88</td>
</tr>
</tbody>
</table>

Table 12. Details regarding the rooms and the meshes used.

<table>
<thead>
<tr>
<th>Room</th>
<th>Auditorium</th>
<th>Concert hall</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dimensions</td>
<td>10m x 10m x 50m</td>
<td>10m x 40m x 50m</td>
</tr>
<tr>
<td>Volume</td>
<td>5 000 m³</td>
<td>20 000 m³</td>
</tr>
<tr>
<td># of elements</td>
<td>1 920k</td>
<td>7 680k</td>
</tr>
<tr>
<td>Schroder Freq.</td>
<td>28 Hz</td>
<td>20 Hz</td>
</tr>
</tbody>
</table>

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 5 000 m³, which is meant to imitate a large auditorium, and the other is 20 000 m³, 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 12.

Table 13. Runtimes in hours for the large shoebox-shaped rooms.

<table>
<thead>
<tr>
<th>#GPUs</th>
<th>Auditorium</th>
<th>Concert hall</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6.52</td>
<td>--</td>
</tr>
<tr>
<td>2</td>
<td>3.35</td>
<td>--</td>
</tr>
<tr>
<td>4</td>
<td>1.82</td>
<td>7.23</td>
</tr>
<tr>
<td>8</td>
<td>1.02</td>
<td>4.41</td>
</tr>
<tr>
<td>12</td>
<td>1.09</td>
<td>3.23</td>
</tr>
</tbody>
</table>

5.6 Complex geometry cases

5.6.1 Auditorium B341-A21 at DTU As an example of using the simulator on realistic 3D geometries, consider the 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/m², 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/m². The initial condition is a Gaussian pulse with width $s_{xy} = 0.3$ m², 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 GPUs results in slightly worse performance compared to the case of using 8 GPUs, indicating that communication costs between cores have begun to dominate. For the concert hall, the simulation cannot be run on 1 or 2 GPUs due to memory limitations. For the higher GPU counts, the runtime drops considerably as the number of GPUs is increased, and likely would 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.
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 $Z = pc = 411.6 \, \text{Ns/m}^2$, 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.

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 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 analysed. 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.

Acknowledgements

The DTU Computing Center (DCC) supported the work with access to state-of-the-art computing resources. The authors wish to thank Mr. Hermes Sampedro Llopis and Mr. Nikolas Borrel-Jensen for fruitful discussions regarding this work.

References


Am. vol 139.4: pp. 1822-1832.


