POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Numerical Solvers for the Second-Order Wave Equation

Numerical Methods for Partial Differential Equations
MSc in High Performance Computing Engineering
Academic Year: 2025-2026

Roberto Di Lauro, 10869236
Thomas Fabbris, 11174296
Alessandro Ghiotto, 11177200
Cristian Rubbi, 11177799

Abstract: This project presents the development, implementation, and performance analysis of a numerical solver for the second-order wave equation. The solver is built upon the **deal.II** finite element library and utilizes **Trilinos** wrappers for distributed linear algebra and parallel computing. The computational domain is discretized using simplicial meshes imported from **GMSH** or generated internally, supporting various polynomial degrees r for the finite element space.

We investigate two main families of time-integration schemes: the **θ-methods** (e.g Forward Euler, Crank-Nicolson, and Backward Euler) and the **Newmark-beta** family . A rigorous numerical analysis is performed, focusing on:

1. **Convergence Analysis**: Evaluation of relative L^2 and H^1 errors across multiple spatial and temporal resolutions to verify theoretical accuracy rates .
2. **Dissipation and Dispersion**: Study of energy conservation properties and phase errors introduced by the discrete approximation .

3. **Scalability**: Performance benchmarks conducted on the **Politecnico di Milano HPC cluster** , evaluating parallel speedup and efficiency.

The results provide a comprehensive characterization of the solver's reliability and efficiency, demonstrating its suitability for large-scale dynamic wave propagation simulations .

1. Introduction

The wave equation is a fundamental mathematical model used to describe the propagation of disturbances in various physical media, spanning from acoustics and electromagnetism to seismology and structural dynamics [?]. In its second-order form, this hyperbolic partial differential equation captures the essential physics of wave motion by balancing temporal acceleration with spatial curvature. Due to its wide range of applications, developing accurate and efficient numerical solvers is crucial for predicting complex dynamic behaviors.

The numerical simulation of wave phenomena, however, introduces significant challenges that go beyond simple stability. While the Courant-Friedrichs-Lewy (CFL) condition [?] provides a standard limit for time-stepping in explicit schemes, the long-term accuracy of the solution is often compromised by numerical dispersion and dissipation [?]. Numerical dispersion leads to frequency-dependent velocity errors that distort the wave shape, while numerical dissipation causes an unphysical decay of the wave's amplitude. Mitigating these errors demands a careful balance between spatial resolution and the choice of the time-integration strategy.

Motivated by these challenges, this work conducts a systematic evaluation of established time-marching strategies [? ?] applied to the semi-discrete wave equation. The primary objective is to move beyond theoretical stability bounds and provide a clear characterization of the practical trade-offs between computational efficiency and physical fidelity in large-scale scenarios. By examining how different algorithmic choices impact energy conservation, phase accuracy, and parallel execution times, this study aims to establish robust guidelines for dynamic wave simulations.

2. Continuous Wave Problem

2.1. Strong Formulation

The propagation of waves in a bounded domain $\Omega \subset \mathbb{R}^2$,as preannounced, is modeled by a second-order hyperbolic Partial Differential Equation [?] . We seek the displacement field $u(\mathbf{x}, t)$ that satisfies the following initial-boundary value problem :

$$\begin{cases} \frac{\partial^2 u}{\partial t^2} - \nabla \cdot (c^2 \nabla u) = f(\mathbf{x}, t) & \text{in } \Omega \times (0, T] \\ u(\mathbf{x}, t) = g(\mathbf{x}, t) & \text{on } \partial\Omega \times [0, T] \\ u(\mathbf{x}, 0) = u_0(\mathbf{x}) & \text{in } \Omega \\ \frac{\partial u}{\partial t}(\mathbf{x}, 0) = v_0(\mathbf{x}) & \text{in } \Omega \end{cases} \tag{1}$$

In this framework, $c(\mathbf{x}, t)$ denotes the wave propagation speed, which is handled in our implementation as a space-time dependent function . The term f represents the external forcing , while u_0 and v_0 define the initial displacement and velocity, respectively. The Dirichlet boundary condition g constrains the solution on the boundary $\partial\Omega$ throughout the simulation interval $[0, T]$.

2.2. Weak Formulation

To apply the Finite Element Method (FEM), we derive the variational form of the strong problem. We begin by defining the appropriate functional spaces for the solution and the test functions. The test space is defined as $V_0 = H_0^1(\Omega) = \{v \in H^1(\Omega) : v|_{\partial\Omega} = 0\}$, which consists of functions with zero trace on the boundary. In the presence of non-homogeneous Dirichlet conditions $g \neq 0$, the solution u belongs to the trial space $V_g = \{v \in H^1(\Omega) : v|_{\partial\Omega} = g\}$.

To ensure compatibility with the Lax-Milgram Lemma [?], which requires trial and test functions to belong to the same Hilbert space, we introduce a **Lifting Operator** $\mathcal{R}$. Let $u_g = \mathcal{R}g \in H^1(\Omega)$ be a function such that its trace on the boundary $\partial\Omega$ is exactly g. We decompose the global solution as:

$$u(\mathbf{x}, t) = u_0(\mathbf{x}, t) + u_g(\mathbf{x}, t) \tag{2}$$

where $u_0 \in V_0$ is the new unknown function with zero trace on the boundary.
The derivation proceeds by multiplying the strong equation by a test function $v \in V_0$ and integrating over the domain Ω:

$$\int_\Omega \frac{\partial^2 u}{\partial t^2} v \, d\Omega - \int_\Omega \nabla \cdot (c^2 \nabla u) v \, d\Omega = \int_\Omega f v \, d\Omega \tag{3}$$

Applying Green's first identity to the second-order spatial term, we obtain:

$$\int_\Omega \frac{\partial^2 u}{\partial t^2} v \, d\Omega + \int_\Omega c^2 \nabla u \cdot \nabla v \, d\Omega - \int_{\partial\Omega} (c^2 \nabla u \cdot \mathbf{n}) v \, d\gamma = \int_\Omega f v \, d\Omega \tag{4}$$

Since $v = 0$ on $\partial\Omega$, the boundary integral vanishes. Substituting the decomposition $u = u_0 + u_g$ and exploiting the linearity of the operators, the semi-discrete weak formulation is expressed in the following way:

$$\begin{cases} \forall t \in (0, T), \text{ find } u_0(t) \in V_0 \text{ such that:} \\ \left(\frac{\partial^2 u_0}{\partial t^2}, v\right)_{L^2(\Omega)} + a(u_0, v) = F(v) - \left(\frac{\partial^2 u_g}{\partial t^2}, v\right)_{L^2(\Omega)} - a(u_g, v) \quad \forall v \in V_0 \end{cases} \tag{5}$$

with the initial conditions:

$$u(x, 0) = u_0(x)$$

$$\frac{\partial u}{\partial t}(x, 0) = u_1(x)$$

This formulation eliminates the second derivatives in space, making the problem compatible with finite H^1 spaces and allowing numerical discretization. Moreover by homogenizing the problem through the lifting operator, the bilinear form a$(\,\cdot\,,\,\cdot\,)$ remains continuous and coercive on the Hilbert space $H_0^1(\Omega)$, guaranteeing the existence and uniqueness of the solution [**?** **?**].

The bilinear and linear forms utilized in the implementation are defined as:

- **Mass form**: $(w, v)_{L^2(\Omega)} = \int_\Omega wv \, d\Omega$(used for assembling the mass matrix $\mathbf{M}$).
- **Stiffness form**: $a(u, v) = \int_\Omega c^2 \nabla u \cdot \nabla v \, d\Omega$(used for assembling the stiffness matrix $\mathbf{A}$).
- **Forcing form**: $F(v) = \int_\Omega f v \, d\Omega$, representing the external source term.

3. Spatial Discretization

To discretize the problem in space using the standard Galerkin method, we introduce a finite-dimensional subspace $V_{h,0} \subset H_0^1(\Omega)$, where $\dim(V_{h,0}) = N_h < +\infty$. In our implementation, this space is constructed using simplicial finite elements of polynomial degree r, specifically through the `FE_SimplexP` class provided by the **deal.II** library [**?**].. The goal is to find an approximation $u_h(t) \in V_{h,0}$ of the exact solution by imposing the weak formulation within the discrete space, leading to the following numerical problem:

$$\begin{cases} \forall t \in (0, T), \text{ find } u_h(t) \in V_{h,0} \text{ such that:} \\ \left(\frac{\partial^2 u_h}{\partial t^2}, v_h\right)_{L^2(\Omega)} + a(u_h, v_h) = F(v_h) \quad \forall v_h \in V_{h,0} \end{cases} \tag{6}$$

To construct an algebraic representation, we consider a basis of $V_{h,0}$ denoted by the shape functions $\{\varphi_j\}_{j=1}^{N_h}$. The discrete solution is expanded as:

$$u_h(\mathbf{x}, t) = \sum_{j=1}^{N_h} U_j(t) \varphi_j(\mathbf{x}) \tag{7}$$

where the coefficients $U_j(t)$ represent the degrees of freedom (DoFs) of the system, collected in the vector $\mathbf{U}(t) = [U_1(t), U_2(t), \ldots, U_{N_h}(t)]^T$. Substituting this expansion into the weak formulation and choosing $v_h = \varphi_i$ as test functions, the linearity of the integral yields:

$$\sum_{j=1}^{N_h} \frac{d^2U_j(t)}{dt^2} \int_\Omega \varphi_j\varphi_i \, d\Omega + \sum_{j=1}^{N_h} U_j(t) \int_\Omega c^2\nabla\varphi_j \cdot \nabla\varphi_i \, d\Omega = \int_\Omega f\varphi_i \, d\Omega \tag{8}$$

This allows us to identify the global matrices assembled by the solver:

- **Mass Matrix M**: $M_{ij} = \int_\Omega \varphi_j\varphi_i \, d\Omega$.
- **Stiffness Matrix A**: $A_{ij} = \int_\Omega c^2\nabla u \cdot \nabla v \, d\Omega$, implemented as $(\nabla\varphi_j, \nabla\varphi_i)$ scaled by the wave speed c^2.
- **Load Vector f**: $f_i = \int_\Omega f\varphi_i \, d\Omega$, representing the external source contribution.

The assembly process is performed in parallel using **Trilinos** distributed matrices. For each active cell in the mesh, local contributions are computed using numerical quadrature, specifically the `QGaussSimplex` rule of order $r + 1$ [**?**]. The resulting semi-discrete dynamical system is:

$$\mathbf{M}\ddot{\mathbf{U}}(t) + \mathbf{A}\mathbf{U}(t) = \mathbf{f}(t) \tag{9}$$

This second-order system of Ordinary Differential Equations represents the spatial discretization(semi-discrete) of the wave equation. In the presence of non-homogeneous Dirichlet conditions, the solver modifies this system during the solution phase to ensure that the degrees of freedom on the boundary correctly reflect the prescribed data $g(\mathbf{x}, t)$.

4. Time Discretization

The spatial discretization detailed in (9) yields a semi-discrete dynamical system subject to the initial conditions $\mathbf{U}(0) = \mathbf{U}_0$ and $\dot{\mathbf{U}}(0) = \mathbf{V}_0$.

To apply standard time-marching techniques, it is common practice to recast (9) as a system of first-order ordinary differential equations [**?**]. We introduce the discrete velocity vector $\mathbf{V}(t) = \dot{\mathbf{U}}(t)$, allowing us to rewrite the problem as two coupled equations:

$$\begin{cases} \dot{\mathbf{U}}(t) - \mathbf{V}(t) = \mathbf{0} \\ \mathbf{M}\dot{\mathbf{V}}(t) + \mathbf{A}\mathbf{U}(t) = \mathbf{F}(t) \end{cases} \tag{10}$$

Boundary conditions are naturally inherited by the velocity field; for a prescribed Dirichlet datum $\mathbf{U}(t) = \mathbf{g}(t)$ on the boundary nodes, the corresponding velocity constraint is strictly enforced as $\mathbf{V}(t) = \dot{\mathbf{g}}(t)$. This coupled first-order system forms the basis for the θ-method family, while the original second-order form (9) is utilized directly by the Newmark-β integration scheme.

4.1. The θ-Method

To discretize the first-order system (10), we adopt a family of implicit-explicit schemes parameterized by $\theta \in [0, 1]$ [**?**]. Time derivatives are approximated using finite differences over a time step Δt, and the equation terms are evaluated as a convex combination of levels t^n and t^{n+1}. The discretized system reads:

$$\frac{\mathbf{U}^{n+1} - \mathbf{U}^n}{\Delta t} = \theta\mathbf{V}^{n+1} + (1 - \theta)\mathbf{V}^n \tag{11}$$

$$\mathbf{M}\frac{\mathbf{V}^{n+1} - \mathbf{V}^n}{\Delta t} + \mathbf{A}(\theta\mathbf{U}^{n+1} + (1 - \theta)\mathbf{U}^n) = \mathbf{F}_\theta^{n+1} \tag{12}$$

where $\mathbf{F}_\theta^{n+1} = \theta\mathbf{F}^{n+1} + (1 - \theta)\mathbf{F}^n$.

By isolating $\mathbf{V}^{n+1}$ in (11) and substituting it into (12), we obtain a single linear system for the unknown displacement $\mathbf{U}^{n+1}$:

$$(\mathbf{M} + \theta^2 \Delta t^2 \mathbf{A})\mathbf{U}^{n+1} = \mathbf{M}(\mathbf{U}^n + \Delta t \mathbf{V}^n) - \theta(1-\theta)\Delta t^2 \mathbf{A}\mathbf{U}^n + \theta \Delta t^2 \mathbf{F}_\theta^{n+1} \tag{13}$$

At each time step, we solve (13) for $\mathbf{U}^{n+1}$. Subsequently, the velocity $\mathbf{V}^{n+1}$ is explicitly updated using (11). The parameter θ dictates the properties of the scheme:

- $\theta = 0$: **Forward Euler**. Fully explicit and conditionally stable.
- $\theta = 1/2$: **Crank-Nicolson**. Implicit, second-order accurate in time, and non-dissipative.
- $\theta = 1$: **Backward Euler**. Fully implicit, generally unconditionally stable, but highly dissipative.

4.2. Newmark Time Integration

The Newmark-β method is a widely used time-marching scheme for second-order dynamical systems [**? ?**]. It computes the state at $t^{n+1} = t^n + \Delta t$ by introducing two parameters, β and γ, which dictate the stability, accuracy, and numerical dissipation.

The discrete displacement $\mathbf{U}$ and velocity $\mathbf{V}$ are updated using the acceleration $\mathbf{a} \approx \ddot{\mathbf{U}}$:

$$\mathbf{U}^{n+1} = \mathbf{U}^n + \Delta t \mathbf{V}^n + \Delta t^2 \left[\left(\frac{1}{2} - \beta \right) \mathbf{a}^n + \beta \mathbf{a}^{n+1} \right] \tag{14}$$

$$\mathbf{V}^{n+1} = \mathbf{V}^n + \Delta t \left[(1-\gamma)\mathbf{a}^n + \gamma \mathbf{a}^{n+1} \right] \tag{15}$$

To compute the unknown acceleration $\mathbf{a}^{n+1}$, we enforce dynamic equilibrium at t^{n+1}:

$$\mathbf{M}\mathbf{a}^{n+1} + \mathbf{A}\mathbf{U}^{n+1} = \mathbf{F}^{n+1} \tag{16}$$

Substituting (14) into this equilibrium equation yields a linear system for $\mathbf{a}^{n+1}$:

$$(\mathbf{M} + \beta \Delta t^2 \mathbf{A})\mathbf{a}^{n+1} = \mathbf{F}^{n+1} - \mathbf{A}\mathbf{U}^n - \Delta t \mathbf{A}\mathbf{V}^n - \Delta t^2 \left(\frac{1}{2} - \beta \right) \mathbf{A}\mathbf{a}^n \tag{17}$$

At each time step, we solve (17) for $\mathbf{a}^{n+1}$, and then explicitly update $\mathbf{U}^{n+1}$ and $\mathbf{V}^{n+1}$. The method's behavior depends heavily on the chosen parameters:

- $\gamma = 1/2, \beta = 1/4$: **Average Acceleration Method**. Unconditionally stable and non-dissipative.
- $\beta > 1/4$: Introduces numerical dissipation, useful for damping high-frequency modes.
- $\gamma < 1/2$: Explicit scheme (conditionally stable).

5. Implementation Details

This section describes the software architecture and organization of the numerical solver for the wave equation.

5.1. Overall Architecture

The software follows an object-oriented design with a modular structure. The implementation is built with C++17 and uses the `deal.II` finite element library for spatial discretization, combined with `Trilinos` for distributed linear algebra operations. The code is designed to support both sequential and parallel (MPI) execution.

5.1.1 Class Hierarchy

The core architecture revolves around a base class and derived implementations:

- **WaveEquationBase**: An abstract base class that encapsulates common functionality for solving the wave equation. This includes mesh management, degree of freedom handling, assembly routines, and I/O operations. The class defines the interface through a pure virtual method `run()` that derived classes must implement.
- **WaveNewmark**: A concrete implementation using the Newmark time integration method. This class inherits from `WaveEquationBase` and implements time-stepping via the Newmark scheme with parameters γ and β. It manages additional data structures specific to the acceleration discretization.
- **WaveTheta**: A concrete implementation using the theta method for time integration. This class also inherits from `WaveEquationBase` and implements time-stepping via a general theta-method scheme parameterized by $\theta \in [0, 1]$.

5.2. Directory Structure

The project is organized as follows:

- **include/**: Contains C++ header files defining the class interfaces. This includes `WaveEquationBase.hpp`, `WaveNewmark.hpp`, `WaveTheta.hpp`, and `ParameterReader.hpp`.
- **src/**: Contains C++ implementation files. The directory holds the implementations of the classes declared in `include/`, as well as the main entry points `main-newmark.cpp` and `main-theta.cpp`.
- **parameters/**: Contains JSON configuration files that specify problem parameters (PDE coefficients, initial/boundary conditions, discretization parameters, etc.). Each JSON file defines a particular test case.
- **mesh/**: Contains mesh files in VTK format used for spatial discretization. Several pre-generated rectangular meshes of different refinement levels are available.
- **results/**: Output directory where solutions and analysis data are stored. Results are organized in subdirectories named after the test case and discretization parameters.
- **analysis/**: Contains Python scripts and Jupyter notebooks for post-processing, convergence analysis, and energy/dissipation studies.
- **report/**: Contains the LaTeX source for the project documentation.

5.3. Parameter System

A key design feature is the flexible parameter input system. The `ParameterReader` class handles parsing of JSON configuration files, which specify:

- **Domain and discretization**: Geometry bounds, number of elements, and polynomial degree r.
- **Temporal parameters**: Final time T, time step size Δt, and method-specific parameters (γ, β for Newmark; θ for theta method).

- **Problem-specific data**: The wave speed $c(\mathbf{x})$, source term $f(t, \mathbf{x})$, initial displacement $u_0(\mathbf{x})$, initial velocity $v_0(\mathbf{x})$, boundary displacement $g(t, \mathbf{x})$, and boundary velocity $\partial_t g(t, \mathbf{x})$.
- **Output control**: Options for saving solutions, logging frequency, and data export settings.

These parameters are specified as mathematical expressions (function strings) parsed by `FunctionParser` from deal.II, allowing for arbitrary analytical expressions without recompilation.

5.4. Execution Flow

The execution of the solver follows this sequence:

1. **Initialization**: A main program (`main-newmark.cpp` or `main-theta.cpp`) is invoked with a path to a JSON parameter file.
2. **Parameter parsing**: The `ParameterReader` reads and validates the JSON file, extracting all problem parameters and function definitions.
3. **Solver instantiation**: A `WaveNewmark` or `WaveTheta` object is created with the parsed parameters.
4. **Setup phase**: The solver's `run()` method is called, which internally:
   - Loads and refines the mesh according to specified parameters.
   - Initializes the finite element space (using simplex elements of degree r).
   - Sets up the degree of freedom handler and distributed numbering (for MPI parallelism).
   - Assembles the mass matrix, stiffness matrix, and initial vectors.
5. **Time integration**: The time-stepping loop advances the solution from $t = 0$ to $t = T$ using either Newmark or theta-method schemes.
6. **Output and logging**: At specified intervals, energy, errors, and solution snapshots are computed and saved to the results directory. Parallel I/O is handled via deal.II's distributed output facilities.

5.5. Parallel Computing Strategy

The code leverages MPI for distributed memory parallelism through deal.II's distributed triangulation and linear algebra. Key aspects include:

- **Mesh distribution**: The mesh is partitioned across MPI processes using `fully_distributed_tria`.
- **DOF distribution**: Degrees of freedom are distributed such that each process owns a portion of unknowns, minimizing communication overhead.
- **Matrix assembly**: Local assembly contributions are computed on each process, then assembled into `TrilinosWrappers::SparseMatrix` objects for efficient distributed sparse linear algebra.
- **Linear algebra**: `Trilinos` provides sparse matrix storage and iterative solvers (Conjugate Gradient for symmetric systems) with distributed preconditioning (AMG).

The MPI communicator and rank information are propagated through the base class, ensuring consistent behavior across all derived implementations.

5.6. Finite Element Discretization

The spatial discretization employs the Galerkin FEM with the following features:

- **Element type**: Simplex (triangular) elements with polynomial degree r (specified per test case).
- **Quadrature**: Standard Gauss quadrature rules are used, with order chosen to exactly integrate polynomial products of the appropriate degree.
- **Matrix assembly**: Assembly routines compute contributions from cell and boundary integrals. The Dirichlet boundary condition (prescribed displacement $g(t, \mathbf{x})$) is imposed after assembly using penalty methods or matrix row modification.
- **Symmetry and definiteness**: Both the mass matrix M and stiffness matrix K are symmetric and positive definite, enabling the use of efficient CG solvers.

5.7. Build System

The project uses CMake for build configuration and compilation. The `CMakeLists.txt` file:

- Detects and configures MPI and deal.II installations.
- Defines compilation targets: a static library `WaveEquationBase` and two executable targets for the Newmark and theta method solvers.
- Sets compiler flags for C++17 standard and enables useful warnings (`-Wfloat-conversion`, `-Wnon-virtual-dtor`, etc.).
- Supports both Release and Debug build types.

5.8. Output and Data Analysis

Simulation outputs are automatically organized in the `results/` directory with subdirectories named by problem identifier and parameters. Output includes:

- **Solution snapshots**: VTK files containing the displacement field $u(\mathbf{x}, t)$ at regular intervals for visualization.
- **Energy data**: CSV files logging the total, kinetic, and potential energy over time for stability and conservation analysis.
- **Error metrics**: Computed difference between the numerical and exact solutions (when an exact solution is available).
- **Performance logs**: Timing information and solver iteration counts.

Post-processing is handled by Python scripts and Jupyter notebooks in the `analysis/` directory, which perform convergence studies, spectral analysis, and visualization.

6. Conclusions

A final section containing the main conclusions of your research/study and possible future developments of your work have to be inserted in the section "Conclusions".