Finite Element Solution of the 2D Wave Equation
Fully Distributed MPI Implementation with θ-Family and Newmark-β

NM-PDE Project

February 25, 2026

1 Introduction and Problem Statement

We solve the wave-equation problem: find $u(x,t)$ such that

$$u_{tt} - c^2 \Delta u = f \quad \text{in } \Omega \times (0, T], \tag{1}$$

with initial conditions

$$u(\cdot, 0) = u_0, \qquad u_t(\cdot, 0) = u_1, \tag{2}$$

and Dirichlet boundary data on $\partial\Omega$.

The implementation is a config-driven MPI C++ solver using deal.II and Trilinos, with simplicial finite elements on triangular Gmsh meshes, two time-discretization families (θ and Newmark-β), ParaView output, and automated convergence studies.

2 Continuous Model and Weak Formulation

2.1 Strong Form and Functional Setting

Let $\Omega \subset \mathbb{R}^2$ be bounded. The weak formulation is posed in Sobolev spaces $H^1(\Omega)$ and $H_0^1(\Omega)$ [2]. For homogeneous Dirichlet data, test functions belong to $V = H_0^1(\Omega)$.

2.2 Homogeneous Dirichlet Weak Form and Semi-Discrete System

For $v \in V$, multiplying by v and integrating by parts gives

$$\langle u_{tt}, v \rangle + c^2 (\nabla u, \nabla v) = (f, v), \qquad \forall v \in V. \tag{3}$$

Using a simplicial FE subspace $V_h = \text{span}\{\varphi_i\}_{i=1}^N$ and $u_h(x,t) = \sum_{j=1}^N U_j(t)\varphi_j(x)$, one obtains the matrix ODE

$$M\ddot{\mathbf{U}}(t) + c^2 K \mathbf{U}(t) = \mathbf{F}(t), \tag{4}$$

with

$$M_{ij} = \int_\Omega \varphi_i \varphi_j \, dx, \quad K_{ij} = \int_\Omega \nabla\varphi_i \cdot \nabla\varphi_j \, dx, \quad F_i(t) = \int_\Omega f(\cdot, t)\varphi_i \, dx. \tag{5}$$

2.3 Non-Homogeneous Dirichlet Data via Lifting

Following the standard lifting argument [1, 2], for boundary data $u = g$ on $\partial\Omega$, write

$$u = w + \widetilde{g}, \qquad w \in H_0^1(\Omega), \qquad \widetilde{g}|_{\partial\Omega} = g. \tag{6}$$

Then w solves

$$\langle w_{tt}, v \rangle + c^2 (\nabla w, \nabla v) = (f, v) - \langle \widetilde{g}_{tt}, v \rangle - c^2 (\nabla\widetilde{g}, \nabla v). \tag{7}$$

In matrix form:

$$M\ddot{\mathbf{W}} + c^2 K\mathbf{W} = \mathbf{F} - M\ddot{\mathbf{G}} - c^2 K\mathbf{G}. \tag{8}$$

In the code, Dirichlet values are imposed strongly at each step on all boundary IDs via constraints and boundary-value elimination. This is algebraically equivalent to working with lifted unknowns and interior DoFs.

3 Space Discretization and Fully Distributed MPI Implementation

The space discretization follows the project requirements and simplicial-FEM theory [1].

3.1 Mesh and FE Space

- Mesh input: Gmsh triangle meshes (`.msh`) generated from `mesh/*.geo`.

- Triangulation: `parallel::fullydistributed::Triangulation<2>`.

- FE: runtime-selectable `FE_SimplexP<2>(degree)` (default degree 1).

- Quadrature: only `QGaussSimplex<2>`.

- DoFs/constraints: `DoFHandler<2>` + hanging-node constraints + Dirichlet BC on all boundary IDs with configured boundary function values.

3.2 Distributed Mesh Import Pipeline

The code uses this pipeline:

1. Read Gmsh mesh into serial `Triangulation<2>` with `GridIn`.

2. Partition with `GridTools::partition_triangulation`.

3. Build a `TriangulationDescription`.

4. Create the fully distributed triangulation from the description.

3.3 Assembly and Linear Algebra

Mass and stiffness matrices are assembled once. The forcing vector $\mathbf{F}(t)$ is assembled each step. All vectors/matrices are distributed Trilinos MPI objects. Implicit systems are solved with Krylov iterations (CG) and AMG-style preconditioning (Trilinos `PreconditionAMG`).

4 Time Discretization

The report peer examples (`2.pdf`, `5.pdf`) emphasized explicit update formulas and stability/dispersion implications. The present report includes those details for both families.

4.1 θ-Method Family

Introduce $\mathbf{v} = \dot{\mathbf{u}}$, giving the first-order system

$$\dot{\mathbf{u}} = \mathbf{v}, \qquad M\dot{\mathbf{v}} + c^2 K\mathbf{u} = \mathbf{F}(t). \tag{9}$$

With step Δt and weight $\theta \in [0, 1]$:

$$\frac{\mathbf{u}^{n+1} - \mathbf{u}^n}{\Delta t} = \theta \mathbf{v}^{n+1} + (1 - \theta)\mathbf{v}^n, \tag{10}$$

$$M\frac{\mathbf{v}^{n+1} - \mathbf{v}^n}{\Delta t} + c^2 K \left(\theta \mathbf{u}^{n+1} + (1 - \theta)\mathbf{u}^n \right) = \mathbf{F}_\theta^{n+1}, \tag{11}$$

where $\mathbf{F}_\theta^{n+1} = \theta \mathbf{F}^{n+1} + (1 - \theta)\mathbf{F}^n$. For $\theta > 0$, eliminating $\mathbf{v}^{n+1}$ gives an SPD displacement solve with effective matrix

$$A_\theta = \frac{1}{\Delta t^2}M + c^2\theta^2 K. \tag{12}$$

Special cases:

- $\theta = 0$: forward Euler-type explicit update (conditionally stable).

- $\theta = 0.5$: Crank–Nicolson (second-order, low dissipation, phase error dominates when Δt is large).

- $\theta = 1$: backward Euler (first-order, strongly dissipative).

4.2 Newmark-β Family

The Newmark update acts directly on $M\ddot{\mathbf{u}} + c^2 K\mathbf{u} = \mathbf{F}$:

$$\mathbf{u}^{n+1} = \mathbf{u}^n + \Delta t\mathbf{v}^n + \Delta t^2 \left(\frac{1}{2} - \beta \right) \mathbf{a}^n + \beta\Delta t^2 \mathbf{a}^{n+1}, \tag{13}$$

$$\mathbf{v}^{n+1} = \mathbf{v}^n + \Delta t(1 - \gamma)\mathbf{a}^n + \Delta t\gamma \mathbf{a}^{n+1}. \tag{14}$$

For $\beta > 0$, the implementation solves for $\mathbf{u}^{n+1}$ through

$$\left(\frac{1}{\beta\Delta t^2}M + c^2 K \right) \mathbf{u}^{n+1} = \mathbf{r}^{n+1}, \tag{15}$$

then reconstructs $\mathbf{a}^{n+1}$ and $\mathbf{v}^{n+1}$. For $\beta = 0$ (central-difference/leapfrog setting with $\gamma = 0.5$), the code uses explicit displacement prediction and a mass solve for acceleration.

Important presets:

- $(\beta, \gamma) = (0.25, 0.5)$ average-acceleration: second-order, unconditionally stable, minimal algorithmic dissipation.

- $(\beta, \gamma) = (0, 0.5)$ central difference style: explicit/conditionally stable, CFL-restricted.

4.3 Dissipation and Dispersion (Concise Discussion)

Consistent with standard references [1, 2] and peer-report observations:

- Strongly implicit first-order damping schemes (e.g., backward Euler) introduce noticeable numerical dissipation.

- Crank–Nicolson and average-acceleration Newmark preserve amplitudes better but can show phase drift (dispersion) when Δt is not small enough.

- Explicit schemes can be low-dissipative in stable CFL regimes, but instability appears quickly once CFL is violated.

5 Software Architecture and Config Workflow

5.1 Source Organization

- `src/wave_solver.*`: FE setup, assembly, MPI vectors/matrices, output, solver interface.

- `src/integrators/theta_integrator.cpp` + `src/theta_integrator.hpp`.

- `src/integrators/newmark_integrator.cpp` + `src/newmark_integrator.hpp`.

- `src/config.*`: strict key=value parser with method-aware validation.

- `src/convergence_runner.*`: spatial/temporal sweep driver and CSV writer.

- `src/function_factory.*`: manufactured solutions / forcing / boundary scenarios.

5.2 CLI and Config

CLI is intentionally minimal:

```
wave-equation --config path/to/file.cfg
```

Key config blocks:

- Core run: `mode`, `method`, `mesh_file`, `fe_degree`, `wave_speed`, `dt`, `n_steps`.

- Output/scenario: `output_interval`, `output_dir`, `scenario_u0`, `scenario_u1`, `scenario_f`, `scenario_bc`.

- Method-specific: `theta` for the θ family; `beta` and `gamma` for Newmark.

- Convergence: `convergence_mesh_files`, `convergence_dt_values`, `convergence_csv_space`, `convergence_csv_time`.

Unknown keys and invalid ranges are rejected during parsing.

6 Convergence Methodology and Results

6.1 Manufactured Reference and Error Norms

To measure error, we use the standing-wave exact solution on $\Omega = [0, 1]^2$:

$$u_{\text{exact}}(x, y, t) = \cos\left(c\sqrt{2}\pi t\right) \sin(\pi x) \sin(\pi y), \tag{16}$$

with compatible initial/boundary data and zero forcing.

At final time t_{f}, errors are computed in:

- L^2 norm,

- full H^1 norm via `VectorTools::H1_norm` (as requested).

6.2 Automated Studies and CSV Output

Implemented modes:

- `convergence_space`: vary mesh hierarchy, fixed Δt.

- `convergence_time`: vary Δt, fixed fine mesh.

- `convergence_both`: run both and write two CSV files.

Each CSV row stores:

```
study,method,fe_degree,theta,beta,gamma,mesh_file,dt,n_steps,t_final,h,ndofs,
l2_error,h1_error,observed_order_l2,observed_order_h1
```

6.3 Observed Behavior from Current Runs

A representative run with implicit Newmark ($\beta = 0.25, \gamma = 0.5$), degree-1 FE, and generated unit-square mesh hierarchy gives:

- Spatial study: observed L^2 orders around $\approx$ 1.7–2.6 and H^1 around $\approx$ 0.87–1.35, consistent with expected $\mathcal{O}(h^2)$ and $\mathcal{O}(h)$ asymptotics for $P1$.

- Temporal study on a fixed mesh: error is nearly saturated (very small slope), indicating spatial error dominates for the selected mesh/time range.

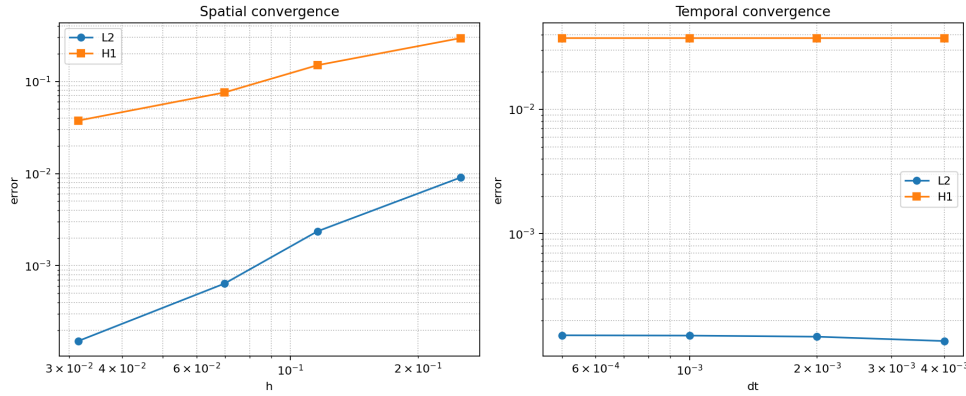 Figure 1 is produced by `scripts/plot_convergence.py` from the generated CSV files.

Figure 1: Log–log convergence plot automatically generated from CSV outputs.

7 How to Build and Run

7.1 Build

```
cmake -S . -B cmake-build-release -DCMAKE_BUILD_TYPE=Release
cmake --build cmake-build-release -j
```

7.2 Mesh Generation

```
./scripts/generate_mesh.sh
```

7.3 Solve Mode

```
mpirun -n 4 ./cmake-build-release/wave-equation \
  --config configs/theta_crank_nicolson.cfg
```

7.4 Convergence + Plot

```
mpirun -n 4 ./cmake-build-release/wave-equation \
  --config configs/convergence_both.cfg
python3 scripts/plot_convergence.py \
  --space-csv solution/convergence_space.csv \
  --time-csv solution/convergence_time.csv \
  --output solution/convergence.png
```

ParaView files (`.pvtu/.vtu`) are written to `solution/`.

8 Conclusions

The project now provides a fully distributed simplicial-FEM wave solver aligned with the requested workflow: separate θ and Newmark integrators, strict config-driven execution, AMG-preconditioned Krylov implicit solves, ParaView export, and automated convergence CSV/plot generation. The report includes full weak-form context, method formulas, and implementation-to-theory mapping.

References

[1] Alfio Quarteroni. *Numerical Models for Differential Problems.* Springer, 2009.

[2] Sandro Salsa. *Partial Differential Equations in Action: From Modelling to Theory.* Springer, 2008.