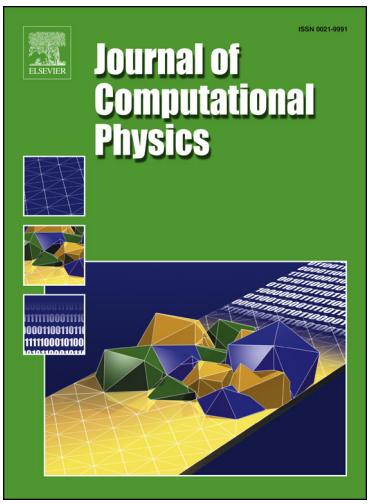
Accepted Manuscript



Newmark local time stepping on high-performance computing architectures

Max Rietmann, Marcus Grote, Daniel Peter, Olaf Schenk

PII: S0021-9991(16)30598-8
DOI: http://dx.doi.org/10.1016/j.jcp.2016.11.012
Reference: YJCPH 6962

To appear in: *Journal of Computational Physics*

Received date: 4 May 2016
Revised date: 30 September 2016
Accepted date: 7 November 2016

Please cite this article in press as: M. Rietmann et al., Newmark local time stepping on high-performance computing architectures, *J. Comput. Phys.* (2016), http://dx.doi.org/10.1016/j.jcp.2016.11.012

Newmark Local Time Stepping on High-Performance Computing Architectures

Max Rietmann[1,3,*], Marcus Grote[2], Daniel Peter[1,3,4], Olaf Schenk[1]

Abstract

In multi-scale complex media, finite element meshes often require areas of local refinement, creating small elements that can dramatically reduce the global time-step for wave-propagation problems due to the CFL condition. Local time stepping (LTS) algorithms allow an explicit time-stepping scheme to adapt the time-step to the element size, allowing near-optimal time-steps everywhere in the mesh. We develop an efficient multilevel LTS-Newmark scheme and implement it in a widely used continuous finite element seismic wave-propagation package. In particular, we extend the standard LTS formulation with adaptations to continuous finite element methods that can be implemented very efficiently with very strong element-size contrasts (more than 100x). Capable of running on large CPU and GPU clusters, we present both synthetic validation examples and large scale, realistic application examples to demonstrate the performance and applicability of the method and implementation on thousands of CPU cores and hundreds of GPUs.

1. Introduction

Efficiently simulating wave propagation at large scales has many important scientific and industrial application domains (for further references see, e.g., [25, 36, 37, 41]). In the field of seismology, simulating seismic waves resulting from an earthquake or other seismic source is an important modality used to better understand the Earth's interior structure and dynamic behavior [11, 38, 40, 42]. Many applications in both forward and inverse modeling

*Please address correspondence to Max Rietmann
Email addresses: max.rietmann@erdw.ethz.ch (Max Rietmann), marcus.grote@unibas.ch (Marcus Grote), daniel.peter@kaust.edu.sa (Daniel Peter), olaf.schenk@usi.ch (Olaf Schenk)
[1]Institute for Computational Science, Università della Svizzera italiana, Lugano, Switzerland
[2]Department of Mathematics and Computer Science, University of Basel, Switzerland
[3]Institute of Geophysics, ETH Zurich, Switzerland
[4]now at: Division of Physical Sciences and Engineering, King Abdullah University of Science and Technology (KAUST), Extreme Computing Research Center (ECRC), Thuwal, Saudi Arabia

have been pushing limits of traditional high-performance computing (HPC) resources for many years [21, 2, 34]. Much of the optimization work in this field is focused on improving the implementation of standard algorithms, which can have bottlenecks that only better algorithm design can remove. Transformative improvements to simulation performance will likely require a coupling of algorithmic, hardware, and software improvements.

In general, the motivating application drives the choice of spatial discretization including a handful of comparable methods, including finite differencing, continuous and discontinuous finite elements, and finite volumes. Finite-element and volume methods are able to use meshes that easily adapt to the spatial domain — some elements can be small where small features are required, and large where large features are needed. However, when a standard explicit time-stepping scheme is used, these small elements require a small time step for stability, enforcing a small time step everywhere in the mesh.

For explicit time-stepping schemes, any local areas of mesh refinement will reduce the global time step thus reducing the efficiency of the method. There are many reasons for local-mesh refinement, but to list a few we see in practice:

1. matching complex external and internal 3D geometry/topography;
2. increased resolution for localized small-scale physics (faults/oceans);
3. mesh-generator difficulties (especially for hexahedral elements, mostly due to sharp angles and/or complicated shapes).

Without a good way to avoid the performance hit of localized refinement, the application scientist usually reduces the scale of the simulation to fit within a computational budget. This algorithmic bottleneck thus often limits scientific work and dictates the computational feasibility of applications. To remedy this shortcoming, we will focus on a method of local time stepping (LTS) that allows the time step to be adapted to the mesh-local spatial resolution.

Previous LTS studies for arbitrary high-order schemes using Adams-Bashforth in time and discontinuous Galerkin (DG) in space (ADER-DG) have been proposed [26] and successfully applied to electromagnetic [39] and elastic wave propagation [9]. These ADER-DG schemes allow each element to take an optimal time-step set by its element-local Courant-Friedrichs-Lewy (CFL) condition. Further work using a DG method done by Gödel et al. [13] was able to show an LTS algorithm working on GPUs for Maxwell's equations. The LTS-leap–frog method proposed by [7] was implemented by [28] using a DG discretization for applications in seismic wave propagation. We note that all of these successful, high-performance implementations of LTS for wave propagation applications have utilized a DG discretization, which may not always be desired. Missing thus far has been an LTS scheme and corresponding high-performance implementation focused on continuous Galerkin finite elements such as the spectral element method (SEM) [19, 22, 23, 27, 30].

We derive an LTS method and its high-performance application of an explicit Newmark time scheme [17], used in some of the most popular community codes in computational seismology [19, 20, 32]. To simplify the development of an LTS variant of the Newmark time-stepping scheme for a SEM, we embrace

the framework developed by Diaz and Grote [7]. They were able to prove and demonstrate optimal convergence and stability properties for second and fourth order leapfrog methods, with recent extension to multiple refinement levels [8]. We derive an LTS variant of the Newmark time-stepping scheme with additional considerations for the SEM, absorbing boundary conditions, and multiple refinement levels.

The structure of the paper is as follows. Section 2 will introduce the two-level LTS-Newmark method, followed by extensions for continuous finite-elements in Section 3, which were required for an efficient implementation. Section 4 extends the two-level scheme to multiple-levels, an important performance feature. In Section 5, we validate the multilevel implementation and introduce the high-performance implementation in the widely used seismic community code SPECFEM3D_Cartesian [5]. Section 6 presents the implementation on massively parallel architectures with details on the load-balancing solution required by a multilevel LTS scheme running on a multinode cluster. Additionally, it presents large-scale synthetic and real-world application benchmarks on CPU and GPU clusters demonstrating the applicability of this new LTS implementation on realistic, large-scale problems where multinode parallelism is a requirement. This is followed by the conclusion in Section 7.

2. Newmark-based Local Time Stepping

Although the LTS-Newmark algorithm can be applied to general wave-propagation problems, we are particularly interested in the elastic wave equation to model seismic wave propagation through the Earth's crust and mantle. The displacement $\vec{u}$ with $x, y,$ and z components satisfies

$$\rho(\vec{x})\vec{u}_{tt} - \nabla \cdot \mathbf{T}(\vec{x}, t) = f(\vec{x}_s, t), \qquad \vec{x} \in \Omega, t > 0, \tag{1}$$

with a traction-free boundary condition with $\hat{\mathbf{r}} \cdot \mathbf{T} = 0$ on the free surface with outward normal $\hat{\mathbf{r}}$. At the vertical and lower boundaries we impose absorbing boundary conditions to keep spurious reflections minimal. The stress tensor $\mathbf{T}(\vec{x}, t)$ is related to the displacement gradient $\nabla\vec{u}$ via Hooke's constitutive law

$$\mathbf{T}(\vec{x}, t) = \mathbf{C}(\vec{x}) : \nabla\vec{u}(\vec{x}, t), \tag{2}$$

where $\mathbf{C}$ is the fourth-order elasticity tensor [6].

As a spatial discretization we utilize a high-order SEM. Following [32], the weak form of (1) – (2) is

$$\int_\Omega \vec{w} \cdot \rho(\vec{x})\vec{u}_{tt}\, d\Omega + \int_\Omega \nabla\vec{w} : \mathbf{T}\, d\Omega = \int_\Omega \vec{w} \cdot f(\vec{x}_s, t)\, d\Omega \quad \forall \vec{w} \in V \tag{3}$$

on the bounded domain Ω, where $V \subset [H^1(\Omega)]^3$ is an appropriately chosen subspace. Given a shape-regular mesh $\mathcal{T}_h$ made of disjoint hexahedral elements K, we let $V_h \subset V$ represent the finite dimensional subspace spanned by the

Lagrangian polynomial basis functions $\vec{\phi}_i = \{\phi_i^x, \phi_i^z, \phi_i^z\}$. Now, we can write (3) in the following matrix form

$$\mathbf{M}\ddot{\mathbf{u}} + \mathbf{K}\mathbf{u} = \mathbf{F}, \tag{4}$$

where

$$\begin{aligned} \mathbf{M}_{ij} &= \int_{\Omega_h} \vec{\phi}_i \cdot \rho_K \vec{\phi}_j \, d\Omega_h, \quad \mathbf{K}_{ij} = \int_{\Omega_h} \nabla\vec{\phi}_i : \mathbf{C}_K : \nabla\vec{\phi}_j \, d\Omega_h, \\ \mathbf{F}_i &= \int_{\Omega_h} \vec{\phi}_i \cdot f(\vec{x}_i, t) \, d\Omega_h, \end{aligned} \tag{5}$$

and we have assumed, for simplicity, that the physical parameters $\rho(\vec{x})$ and $\mathbf{C}(\vec{x})$ are constant within each element K. The degrees of freedom (DOF) $\mathbf{u}_i$ are ordered by nodal values as

$$\mathbf{u} = [u^x(\vec{x}_1), u^y(\vec{x}_1), u^z(\vec{x}_1), \ldots, u^x(\vec{x}_N), u^y(\vec{x}_N), u^z(\vec{x}_N)] \tag{6}$$

The choice of Gauss-Lobatto-Legendre (GLL) collocation points in each element K combined with the appropriate quadrature leads to a diagonal matrix $\mathbf{M}$ without loss of accuracy [19]. Thus, (4) can be rewritten in a form that allows for an explicit time-stepping scheme

$$\ddot{\mathbf{u}} = -\mathbf{M}^{-1}(\mathbf{K}\mathbf{u} - \mathbf{F}) = \mathbf{B}\mathbf{u} + \tilde{\mathbf{F}}, \tag{7}$$

as $\mathbf{M}^{-1}$ is computed trivially; hence, $\mathbf{B}$ fully represents the spatial discretization. Each x, y, z triplet in $\mathbf{u}$ represents a GLL point (node) on each element, where nodes on element-boundaries are shared between elements. To finalize the discretization of (7), we must choose a time-stepping method.

2.1. Newmark method

The explicit Newmark scheme is a relatively popular method currently used in several spectral element implementations [20, 29, 10, 32] that is second order accurate and conserves an equivalent energy [24] and is equivalent to the second-order leap–frog scheme — see Section 2.4. To derive an LTS version of Newmark, we will first rederive the standard Newmark scheme from first principles. Starting from (7), with $\tilde{\mathbf{F}} = 0$, for simplicity, we rewrite it as the first-order system,

$$\begin{aligned} \dot{\mathbf{u}} &= \mathbf{v}, \\ \dot{\mathbf{v}} &= \mathbf{B}\mathbf{u}. \end{aligned} \tag{8}$$

The solution of (8) is formally given by

$$\begin{aligned} \mathbf{u}(t_n + \xi\Delta t) &= \mathbf{u}(t_n) + \int_{t_n}^{t_n+\xi\Delta t} \mathbf{v}(s) \, ds, \\ \mathbf{v}(t_n + \xi\Delta t) &= \mathbf{v}(t_n) + \int_{t_n}^{t_n+\xi\Delta t} \mathbf{B}\mathbf{u}(s) \, ds. \end{aligned} \tag{9}$$

Any numerical scheme must approximate the integrands $\mathbf{v}(s)$ and $\mathbf{Bu}(s)$ in (9). In deriving the Newmark scheme from this integral formulation, we advance $\mathbf{u}(t)$ and $\mathbf{v}(t)$ on staggered temporal grids, whereby we can utilize the same midpoint quadrature rule for both integrands:

$$\begin{aligned} \mathbf{v}(t_n + \tfrac{1}{2}\Delta t) &= \mathbf{v}(t_n - \tfrac{1}{2}\Delta t) + \int_{t_n - \frac{1}{2}\Delta t}^{t_n + \frac{1}{2}\Delta t} \mathbf{Bu}(s)\, ds \\ &\approx \mathbf{v}(t_n - \tfrac{1}{2}\Delta t) + \Delta t\, \mathbf{Bu}(t_n), \\ \mathbf{u}(t_n + \Delta t) &= \mathbf{u}(t_n) + \int_{t_n}^{t_n + \Delta t} \mathbf{v}(s)\, ds \\ &\approx \mathbf{u}(t_n) + \Delta t\, \mathbf{v}(t_n + \tfrac{1}{2}\Delta t). \end{aligned} \tag{10}$$

Next, we denote by $\mathbf{v}_{n+\xi}$ the numerical approximation at time $t_n + \xi\Delta t$, and thus write the above time-marching scheme succinctly as

$$\begin{aligned} \mathbf{v}_{n+\frac{1}{2}} &= \mathbf{v}_{n-\frac{1}{2}} + \Delta t\, \mathbf{Bu}_n, \\ \mathbf{u}_{n+1} &= \mathbf{u}_n + \Delta t\, \mathbf{v}_{n+\frac{1}{2}}. \end{aligned} \tag{11}$$

This staggered form will be useful in the derivation of LTS-Newmark.

A more commonly written form takes advantage of an intermediate variable $\mathbf{a}_n$, such that

$$\mathbf{a}_n = \mathbf{Bu}_n, \quad \mathbf{v}_n = \mathbf{v}_{n-\frac{1}{2}} + \tfrac{1}{2}\Delta t\, \mathbf{a}_n. \tag{12}$$

Indeed, we can then rewrite $\mathbf{u}_{n+1}$ as

$$\begin{aligned} \mathbf{u}_{n+1} &= \mathbf{u}_n + \Delta t \mathbf{v}_{n-\frac{1}{2}} + \tfrac{1}{2}\Delta t^2 \mathbf{Bu}_n + \tfrac{1}{2}\Delta t^2 \mathbf{Bu}_n \\ &= \mathbf{u}_n + \Delta t \mathbf{v}_n + \tfrac{1}{2}\Delta t^2 \mathbf{a}_n. \end{aligned} \tag{13}$$

Similarly, we can use (11) and (12) to rewrite $\mathbf{v}_{n+1}$ as

$$\begin{aligned} \mathbf{v}_{n+1} &= \mathbf{v}_{n+\frac{1}{2}} + \tfrac{1}{2}\Delta t\, \mathbf{a}_{n+1} \\ &= \mathbf{v}_{n-\frac{1}{2}} + \tfrac{1}{2}\Delta t\, \mathbf{a}_n + \tfrac{1}{2}\Delta t\, \mathbf{a}_n + \tfrac{1}{2}\Delta t\, \mathbf{a}_{n+1} \\ &= \mathbf{v}_n + \tfrac{1}{2}\Delta t\, \mathbf{a}_n + \tfrac{1}{2}\Delta t\, \mathbf{a}_{n+1}, \end{aligned} \tag{14}$$

which yields the more familiar (non-staggered) form [17],

$$\begin{aligned} \mathbf{u}_{n+1} &= \mathbf{u}_n + \Delta t\, \mathbf{v}_n + \frac{\Delta t^2}{2} \mathbf{a}_n, \\ \mathbf{a}_{n+1} &= \mathbf{Bu}_{n+1} + \tilde{\mathbf{F}}_{n+1}, \\ \mathbf{v}_{n+1} &= \mathbf{v}_n + \frac{\Delta t}{2} \left(\mathbf{a}_n + \mathbf{a}_{n+1} \right), \end{aligned} \tag{15}$$

Regarding the conservative properties of Newmark, we note that the semi-discrete form (4) conserves the semi-discrete energy [24]

$$E = \tfrac{1}{2} \dot{\mathbf{u}}^T \mathbf{M} \dot{\mathbf{u}} + \tfrac{1}{2} \mathbf{u}^T \mathbf{K} \mathbf{u}, \tag{16}$$

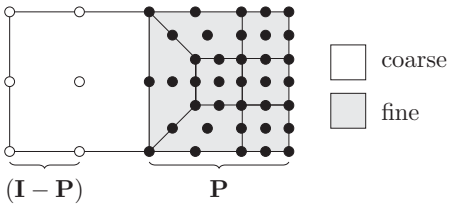


Figure 1: 2D example mesh with refinement with second-order elements (9 nodes per element). Coarse-region $(\mathbf{I}-\mathbf{P})$ nodes marked in white-filled circles and fine-region $(\mathbf{P})$ nodes marked in black-filled circles.

if $\tilde{\mathbf{F}} = 0$. Implicit forms of Newmark conserve this quantity, however the explicit form of Newmark considered here conserves a modified energy,

$$E_{\mathrm{NM}} = \tfrac{1}{2}\mathbf{v}^T\mathbf{M}\mathbf{v} + \tfrac{1}{2}\mathbf{u}^T(\mathbf{K} - \tfrac{1}{4}\Delta t^2\mathbf{K}\mathbf{M}^{-1}\mathbf{K})\mathbf{u}, \tag{17}$$

and in the limit $\Delta t \to 0$ is equal to the semi-discrete form (16). To ensure the positive definiteness of $(\mathbf{K} - \frac{1}{4}\Delta t^2\mathbf{K}\mathbf{M}^{-1}\mathbf{K})$ and thereby the stability of the scheme, the time step Δt must be chosen to be sufficiently small, and generally follows the commonly known CFL condition. For acoustic or elastic (1) wave propagation it is the following inequality,

$$\Delta t \leq C_{\mathrm{CFL}} \min_{K_i \in \mathcal{T}_h} \left(\tilde{h}_i\right), \qquad \tilde{h}_i = \frac{h_i}{c_i}, \tag{18}$$

where h_i and c_i denote the element size (or radius) and material velocity inside element $K_i \in \mathcal{T}_h$. The element size h_i will be dependent on the element type, and there are many choices of size metric for a variety of element types in 2D and 3D, including shortest element edge or radius of inscribed circle, such that the choice will affect the final CFL constant C_{CFL}. Due to this stability condition, the smallest $\tilde{h}_i$ will impose the same (small) Δt everywhere in the mesh. To overcome that bottleneck, we will now derive an LTS method, which allows differently sized time steps in different regions of the mesh.

2.2. LTS-Newmark method

Following the framework in [7, 14, 15], we divide the finite-element mesh into both *fine* and *coarse* element regions and correspondingly we split the DOFs as

$$\mathbf{u}(t) = \mathbf{P}\mathbf{u}(t) + (\mathbf{I}-\mathbf{P})\mathbf{u}(t) = \mathbf{u}^{[\mathrm{fine}]}(t) + \mathbf{u}^{[\mathrm{coarse}]}(t). \tag{19}$$

Here the selection matrix $\mathbf{P}$ is diagonal: $\mathbf{P}_{ii} = 1$ when the associated DOF u_i is within the fine region, and zero elsewhere. In a continuous finite element method, such as a SEM, the wavefield $\mathbf{u}(t)$ is defined on mesh nodes that are shared on element boundaries. To resolve the ambiguity of element-boundary nodes, we assume that all coarse-fine boundary nodes belong to the fine region, which has been depicted in the 2D example in Fig. 1. Next we approximate, for

fixed t_n, the integrand in (9) as

$$\mathbf{B}((\mathbf{I}-\mathbf{P})\mathbf{u}(t)+\mathbf{P}\mathbf{u}(t)) \approx \mathbf{B}((\mathbf{I}-\mathbf{P})\mathbf{u}(t_n)+\mathbf{P}\tilde{\mathbf{u}}(\tau)), \tag{20}$$

where $\tilde{\mathbf{u}}(\tau)$ solves the differential equation

$$\begin{aligned}
\frac{d\tilde{\mathbf{u}}}{d\tau}(\tau) &= \tilde{\mathbf{v}}(\tau), \quad -\Delta t < \tau < \Delta t, \qquad \tilde{\mathbf{u}}(0) = \mathbf{u}(t_n) \\
\frac{d\tilde{\mathbf{v}}}{d\tau}(\tau) &= \mathbf{B}(\mathbf{I}-\mathbf{P})\mathbf{u}(t_n) + \mathbf{B}\mathbf{P}\tilde{\mathbf{u}}(\tau), \qquad \tilde{\mathbf{v}}(0) = \nu
\end{aligned} \tag{21}$$

where ν will be specified below.

This "fine-level" subproblem allows us to advance the fine-region values at a finer time step that satisfies the local CFL condition and frees the coarse-level elements to take larger steps. Note that $\mathbf{u}(t_n)$ is a constant within this subproblem.

Using (20) we now approximate the solution of our original system (8) as:

$$\begin{aligned}
\mathbf{v}(t_n+\tfrac{1}{2}\Delta t) &\approx \mathbf{v}(t_n-\tfrac{1}{2}\Delta t) + \int_{-\frac{1}{2}\Delta t}^{\frac{1}{2}\Delta t} \mathbf{B}(\mathbf{I}-\mathbf{P})\mathbf{u}(t_n) + \mathbf{B}\mathbf{P}\tilde{\mathbf{u}}(s)\,ds \\
&\approx \mathbf{v}(t_n-\tfrac{1}{2}\Delta t) + \Delta t\,\mathbf{B}(\mathbf{I}-\mathbf{P})\mathbf{u}(t_n) + \int_{-\frac{1}{2}\Delta t}^{\frac{1}{2}\Delta t} \mathbf{B}\mathbf{P}\tilde{\mathbf{u}}(s)\,ds, \\
\mathbf{u}(t_n+\Delta t) &\approx \mathbf{u}(t_n) + \int_{t_n}^{t_n+\Delta t} \mathbf{v}(s)|_{s=t_{n+\frac{1}{2}}}\,ds \\
&\approx \mathbf{u}(t_n) + \Delta t\,\mathbf{v}_{n+\frac{1}{2}}.
\end{aligned} \tag{22}$$

Since $\mathbf{u}(t_n)$ is constant within (21), we have

$$\tilde{\mathbf{v}}(\tfrac{1}{2}\Delta t) = \tilde{\mathbf{v}}(-\tfrac{1}{2}\Delta t) + \Delta t\,\mathbf{B}(\mathbf{I}-\mathbf{P})\mathbf{u}(t_n) + \int_{-\frac{1}{2}\Delta t}^{\frac{1}{2}\Delta t} \mathbf{B}\mathbf{P}\tilde{\mathbf{u}}(s)\,ds, \tag{23}$$

$$\begin{aligned}
\tilde{\mathbf{u}}(\Delta t) &= \mathbf{u}(t_n) + \int_0^{\Delta t} \tilde{\mathbf{v}}(s)\,ds \\
&\approx \mathbf{u}(t_n) + \tilde{\mathbf{v}}(\tfrac{1}{2}\Delta t).
\end{aligned} \tag{24}$$

A comparison of (22) with (23) yields

$$\mathbf{v}(t_n+\tfrac{1}{2}\Delta t) - \mathbf{v}(t_n-\tfrac{1}{2}\Delta t) \approx \tilde{\mathbf{v}}(\tfrac{1}{2}\Delta t) - \tilde{\mathbf{v}}(-\tfrac{1}{2}\Delta t). \tag{25}$$

Since $\tilde{\mathbf{v}}(\tau) - \tilde{\mathbf{v}}(-\tau)$ is independent of ν, we may choose $\nu = 0$ in (21), which implies that $\tilde{\mathbf{u}}(\tau) = \tilde{\mathbf{u}}(-\tau)$ and $\tilde{\mathbf{v}}(\tau) = -\tilde{\mathbf{v}}(-\tau)$. Hence, we can rewrite (25) as

$$\begin{aligned}
\mathbf{v}(t_n+\tfrac{1}{2}\Delta t) &\simeq \mathbf{v}(t_n-\tfrac{1}{2}\Delta t) + 2\,\tilde{\mathbf{v}}(\tfrac{1}{2}\Delta t) \\
&\simeq \mathbf{v}(t_n-\tfrac{1}{2}\Delta t) + 2\left(\frac{\tilde{\mathbf{u}}(\Delta t) - \mathbf{u}(t_n)}{\Delta t}\right)
\end{aligned} \tag{26}$$

where $\tilde{\mathbf{v}}(\frac{1}{2}\Delta t)$ is replaced by the approximation in (24) and $\tilde{\mathbf{u}}(\Delta t)$ is computed by applying Newmark to (21) with a smaller and stable time step $\Delta\tau$.

Depending on the relative size of elements in the coarse region, the LTS-Newmark algorithm will use the maximum stable integer multiple time step $\Delta t = p\,\Delta\tau$ such that

$$p = \left\lfloor \frac{\tilde{h}_{\min}^{[\text{coarse}]}}{\tilde{h}_{\min}^{[\text{fine}]}} \right\rfloor \in \mathbb{N}. \tag{27}$$

The scheme thus takes p steps of size $\Delta\tau = \frac{\Delta t}{p}$ in the fine region for every larger Δt step in the coarse region.

2.3. LTS-algorithm for two levels

With the expression in (26) defining the updates on the global level, we can write down the LTS-Newmark scheme for two levels in Algorithm 1. The use of $\mathbf{w}$ avoids the re-computation of $\mathbf{B}(\mathbf{I}-\mathbf{P})\mathbf{u}_n$ at every substep, ideally saving a significant amount of work. All further matrix-vector multiplications by $\mathbf{BP}$ involve only those DOFs associated with the fine region, which take a smaller time step $\Delta\tau$ imposed by the local CFL condition.

Algorithm 1 Two-level LTS-Newmark

Require: $\mathbf{u}_0, \mathbf{v}_{-\frac{1}{2}}, \Delta\tau = \frac{\Delta t}{p}$
  for $n = 0, 1, \ldots$ **do**
    $\mathbf{w} = \mathbf{B}(\mathbf{I}-\mathbf{P})\mathbf{u}_n$
    $\tilde{\mathbf{v}}_{\frac{1}{2}} = \frac{1}{2}\Delta\tau(\mathbf{BP}\mathbf{u}_n + \mathbf{w})$
    $\tilde{\mathbf{u}}_1 = \mathbf{u}_n + \Delta\tau\tilde{\mathbf{v}}_{\frac{1}{2}}$
    for $m = 1, \ldots, (p-1)$ **do**
      $\tilde{\mathbf{v}}_{m+\frac{1}{2}} = \tilde{\mathbf{v}}_{m-\frac{1}{2}} + \Delta\tau\mathbf{w} + \Delta\tau\mathbf{BP}\tilde{\mathbf{u}}_m$
      $\tilde{\mathbf{u}}_{m+1} = \tilde{\mathbf{u}}_m + \Delta\tau\,\tilde{\mathbf{v}}_{m+\frac{1}{2}}$
    end for
    $\mathbf{v}_{n+\frac{1}{2}} = \mathbf{v}_{n-\frac{1}{2}} + 2\left(\frac{\tilde{\mathbf{u}}_p - \mathbf{u}_n}{\Delta t}\right)$
    $\mathbf{u}_{n+1} = \mathbf{u}_n + \Delta t\mathbf{v}_{n+\frac{1}{2}}$
  end for

If the coarse region has relatively more elements than the fine region, LTS will be able to save a large amount of computation, that can be modeled simply as

$$\text{theoretical LTS speedup} = \frac{p \times \#[\text{fine + coarse elements}]}{p \times \#[\text{fine elements}] + \#[\text{coarse elements}]}, \tag{28}$$

where we note that each fine element has to do p-times more work than a coarse element to reach the final desired elapsed time. Note that for $p = 1$, Algorithm 1 coincides with the standard Newmark method (11).

Remark: Many applications incorporate some form of damping, be it physical or due to first-order absorbing boundary conditions (ABC) [4], yielding the

following discrete system

$$\mathbf{M}\ddot{\mathbf{u}} + \mathbf{C}\dot{\mathbf{u}} + \mathbf{K}\mathbf{u} = 0,$$

similar to (4) (with $\mathbf{F} = 0$), however, with the diagonal matrix $\mathbf{C}$ representing damping or a boundary term.

The addition of a term including $\dot{\mathbf{u}}$ requires certain changes to the time-stepping scheme. For the staggered non-LTS Newmark scheme (11), we have

$$\begin{aligned} \mathbf{v}_{n+\frac{1}{2}} &= \mathbf{v}_{n-\frac{1}{2}} - \Delta t\, \tilde{\mathbf{M}}^{-1} \left(\mathbf{K}\mathbf{u}_n + \mathbf{C}\mathbf{v}_{n-\frac{1}{2}} \right), \\ \mathbf{u}_n &= \mathbf{u}_n + \Delta t\, \mathbf{v}_{n+\frac{1}{2}}, \end{aligned}$$

where $\tilde{\mathbf{M}} = \mathbf{M} + \frac{1}{2}\Delta t\mathbf{C}$. This formulation allows us to modify the two-level LTS-Newmark scheme (Algorithm 1) by simply using the modified mass matrix applied to the spatial operator $\tilde{\mathbf{B}} = -\tilde{\mathbf{M}}^{-1}\mathbf{K}$ and a modified global update

$$\mathbf{v}_{n+\frac{1}{2}} = \mathbf{v}_{n-\frac{1}{2}} + 2\left(\frac{\tilde{\mathbf{u}}_p - \mathbf{u}_n}{\Delta t} \right) - \Delta t \tilde{\mathbf{M}}^{-1}\mathbf{C}\mathbf{v}_{n-\frac{1}{2}},$$

as the only changes.

When the absorbing boundary is made of only coarse-region elements, there is no effect on stability or the qualitative nature of the absorption. However, when fine-region elements are included in the absorbing boundary, the stability of the scheme is impacted. In 1-D acoustic experiments, the largest stable time step was reduced by over a third, potentially reducing the effectiveness of LTS in practice. Further study of this stability reduction and replicating this result with other LTS schemes goes beyond the scope of this paper, but would be interesting future work.

2.4. Accuracy, convergence and stability analysis

To rigorously establish the accuracy and conservative properties of our new LTS-Newmark scheme from Section 2.2, we will rewrite it in a single-step "Newmark" formulation, which first requires the following technical result:

Lemma 2.1. *For $p \geq 1$, $\tilde{\mathbf{u}}_j$ defined by Algorithm 1 satisfies*

$$\tilde{\mathbf{u}}_j = \mathbf{u}_n + \sum_{i=0}^{j-1} \alpha_i^j (\Delta\tau)^{2i+2} (\mathbf{B}\mathbf{P})^i \mathbf{B}\mathbf{u}_n, \quad 1 \leq j \leq p, \tag{29}$$

where the constants α_i^j, $0 \leq i \leq j-1$, are defined by the following recurrences:

$$\begin{cases} \alpha_0^1 = \frac{1}{2}, \quad \alpha_0^2 = 2, \quad \alpha_1^2 = \frac{1}{2}, \\ \alpha_0^j = \frac{j^2}{2}, \\ \alpha_i^j = 2\alpha_i^{j-1} - \alpha_i^{j-2} + \alpha_{i-1}^{j-1}, \quad i = 1, \ldots, j-3, \\ \alpha_{j-2}^j = 2\alpha_{j-2}^{j-1} + \alpha_{j-3}^{j-1}, \\ \alpha_{j-1}^j = \alpha_{j-2}^{j-1}. \end{cases} \tag{30}$$

The proof is provided in Appendix Appendix A. Therefore, we can rewrite the LTS algorithm in "Newmark manner".

Proposition 1. *For $p \geq 2$, Algorithm 1 is equivalent to*

$$\begin{aligned} \mathbf{v}_{n+\frac{1}{2}} &= \mathbf{v}_{n-\frac{1}{2}} + \Delta t\, \mathbf{B}_p \mathbf{u}_n, \\ \mathbf{u}_{n+1} &= \mathbf{u}_n + \Delta t\, \mathbf{v}_{n+\frac{1}{2}}, \end{aligned}$$

where $\mathbf{B}_p$ is defined as

$$\mathbf{B}_p = \mathbf{B} + \frac{2}{p^2} \sum_{i=1}^{p-1} \alpha_i^p (\Delta\tau)^{2i} (\mathbf{BP})^i \mathbf{B}. \tag{31}$$

and the constants α_i^p are given by (30).

Proof: We rewrite $\mathbf{v}_{n+\frac{1}{2}}$ from Algorithm 1 using Lemma 2.1 as

$$\mathbf{v}_{n+\frac{1}{2}} = \mathbf{v}_{n-\frac{1}{2}} + \frac{2}{\Delta t}\left(\tilde{\mathbf{u}}_p - \mathbf{u}_n\right) = \mathbf{v}_{n-\frac{1}{2}} + \frac{2}{\Delta t}\left(\sum_{i=0}^{p-1} \alpha_i^p (\Delta\tau)^{2i+2} (\mathbf{BP})^i \mathbf{B}\mathbf{u}_n\right).$$

After pulling out the first term in the sum and using $\Delta\tau = \Delta t/p$ and $\alpha_0^p = p^2/2$, we obtain

$$\mathbf{v}_{n+\frac{1}{2}} = \mathbf{v}_{n-\frac{1}{2}} + \Delta t\left(\mathbf{B} + \frac{2}{p^2} \sum_{i=1}^{p-1} \alpha_i^p (\Delta\tau)^{2i} (\mathbf{BP})^i \mathbf{B}\right) \mathbf{u}_n,$$

which completes the proof. □

As a consequence of Prop. 1, we can also rewrite LTS-Newmark in "leap–frog manner" as

$$\mathbf{u}_{n+1} = 2\mathbf{u}_n - \mathbf{u}_{n-1} + \Delta t^2\, \mathbf{B}_p \mathbf{u}_n, \tag{32}$$

which corresponds to the LTS-leap–frog method from [7] — Proposition 1 coincides with (Proposition 3.3, [7]) with $\mathbf{B} = -\mathbf{A}, \mathbf{B}_p = -\mathbf{A}_p$ and α_i^p premultiplied by $(-1)^{i+1}$. Hence, the current LTS-Newmark formulation immediately inherits the properties proved in [7]. In particular, it is also second-order accurate and conserves a discrete version of the energy like the standard (non-LTS) Newmark method.

3. LTS-Newmark Formulation for a Continuous FEM

The principle focus of this paper is the derivation and implementation of a high-performance LTS-Newmark method. The previously defined algorithm does not make explicit considerations for an efficient implementation with continuous finite elements that can achieve the LTS speedup predicted by (28).

Algorithm 1 does explicitly provide the precomputation of $\mathbf{w} = \mathbf{B}(\mathbf{I} - \mathbf{P})\mathbf{u}_n$, which ensures that this expensive operation is only done once per global time

step. However it is unclear how to avoid computing updates on coarse-region nodes $(\mathbf{I}-\mathbf{P})$ in a substep (e.g., $\tilde{\mathbf{u}}_{m+1} = \tilde{\mathbf{u}}_m + \Delta\tau\tilde{\mathbf{v}}_{m+\frac{1}{2}}$). To better understand this, we list the vector additions of LTS-Newmark and the actual and optimal computational cost in terms of the coarse and fine DOFs. If we let N_{coarse} and N_{fine} represent the number of coarse and fine elements, one can write the computational complexity of the operations from Algorithm 1 in terms of their actual cost and the optimal cost if we consider LTS as an algorithm acting on each refinement-level locally.

	Operation	Actual Cost	Optimal Cost
(1)	$\mathbf{w} = \mathbf{B}(\mathbf{I}-\mathbf{P})\mathbf{u}_n$	$\mathrm{O}(N_{\text{coarse}})$	$\mathrm{O}(N_{\text{coarse}})$
(2)	$\tilde{\mathbf{a}} = \mathbf{BP}\tilde{\mathbf{u}}_m$	$\mathrm{O}(N_{\text{fine}})$	$\mathrm{O}(N_{\text{fine}})$
(3)	$\tilde{\mathbf{v}}_{m+\frac{1}{2}} = \tilde{\mathbf{v}}_{m-\frac{1}{2}} + \Delta\tau\mathbf{w} + \Delta\tau\tilde{\mathbf{a}}$	$\mathrm{O}(N_{\text{coarse}} + N_{\text{fine}})$	$\mathrm{O}(N_{\text{fine}})$
(4)	$\tilde{\mathbf{u}}_{m+1} = \tilde{\mathbf{u}}_m + \Delta\tau\tilde{\mathbf{v}}_{m+\frac{1}{2}}$	$\mathrm{O}(N_{\text{coarse}} + N_{\text{fine}})$	$\mathrm{O}(N_{\text{fine}})$
(5)	$2\left(\frac{\tilde{\mathbf{u}}_p - \mathbf{u}_n}{\Delta t}\right)$	$\mathrm{O}(N_{\text{coarse}} + N_{\text{fine}})$	$\mathrm{O}(N_{\text{fine}})$

Table 1: Actual vs. optimal LTS-Newmark operations as listed in Algorithm 1.

Several operations from Algorithm 1 are listed in Table 1, along with their actual and optimal computational costs. For example, the $\mathbf{B}(\mathbf{I}-\mathbf{P})\mathbf{u}$ and $\mathbf{BP}\tilde{\mathbf{u}}$ operations on the coarse and fine region already compute the minimal set of operations necessary, with the reasonable assumption that $\mathbf{Pu}$ and $(\mathbf{I}-\mathbf{P})\mathbf{u}$ can be implemented efficiently. The operation $\mathbf{Bu}$ is typically done in a matrix-free fashion with a loop over elements, and by restricting the loop to act only on elements with non-zero $\mathbf{P}$ or $(\mathbf{I}-\mathbf{P})$, the optimal complexity can be achieved directly. However, the vector operations (3), (4), and (5) are done on the full set of nodes and it is not clear how to select the optimal set of nodes, which are required for the algorithm, in contrast to operations (1) and (2). To achieve very high efficiency, we have to extract the minimal set of nodes in $\mathbf{P}$ and $(\mathbf{I}-\mathbf{P})$ required to initialize, execute, and finalize the fine-region substeps, such that fine-region operations have a cost of $\mathrm{O}(N_{\text{fine}})$. Thus, we need to carefully consider the boundaries between coarse and fine elements in the context of a continuous FEM.

The spatial discretization operator $\mathbf{B}$ mixes information across element boundaries due to the continuous nature of the finite-element basis functions. To characterize this mixing in the discretized system (7), we define two further selection matrices:

Coarse-to-Fine contributions: Selection matrix $\mathbf{R}$ is a diagonal matrix with 1 at *coarse* nodes that are in an element also containing fine nodes and zero everywhere else, with properties

$$\mathbf{PBR} \neq \mathbf{0}, \quad \mathbf{PB}(\mathbf{I}-\mathbf{P}-\mathbf{R}) = \mathbf{0}. \tag{33}$$

In other words, $\mathbf{R}$ selects coarse nodes that, through $\mathbf{B}$, contribute to the fine region.

Fine-to-Coarse contributions: Selection matrix $\mathbf{F}$ is a diagonal matrix with 1 at *fine* nodes that are directly bordering coarse nodes and zero everywhere else, with properties

$$(\mathbf{I}-\mathbf{P})\mathbf{B}\mathbf{F} \neq \mathbf{0}, \quad (\mathbf{I}-\mathbf{P})\mathbf{B}(\mathbf{P}-\mathbf{F}) = \mathbf{0}. \tag{34}$$

In other words, $\mathbf{F}$ selects fine nodes that, through $\mathbf{B}$, contribute to the coarse region.

These additional selection matrices are illustrated in Figure 2 with the continuous nodal basis functions that define the DOFs.

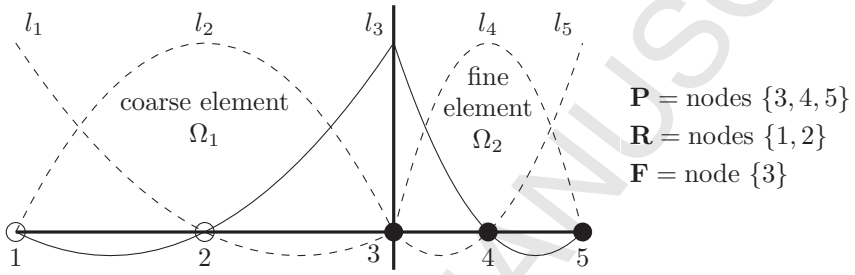


Figure 2: A 1D example of the interface between two refinement levels from the perspective of a fine element. The $l_i(x)$ represent a second-order polynomial finite-element basis set on the GLL points (for $N=2$). $l_3(x)$, highlighted with a solid line, has support over both elements, whereas the $l_{i\neq 3}$, marked with dotted lines, only have support in their respective elements. Intuitively, the spatial operator $\mathbf{B}$ smears values at nodes 1 and 2 into the neighboring fine element via node 3, and vice versa.

In order to implement LTS-Newmark for a SEM efficiently, the use of the $\mathbf{P}, \mathbf{R}$, and $\mathbf{F}$ selection matrices is needed so that the implementation only updates the substep $\Delta\tau$ values of coarse nodes where the fine region requires their value. Hence, these matrices help define the flow of information across element boundaries, a crucial element of LTS.

In contrast, other finite-element formulations such as DG [16], can directly implement LTS without explicitly using these local "communication" matrices $\mathbf{R}$ and $\mathbf{F}$, thereby easing high-performance LTS implementations [9, 13]. As the element boundaries, and therefore the refinement-level boundaries, are explicitly coupled via the numerical flux, implementing LTS from the global DG formulation can be relatively straightforward. As noted in [15], the choice of spatial discretization does not impact the convergence or stability properties of the LTS method. Given our desire to use a SEM with its continuous nodal basis, we examine the terms $\mathbf{B}(\mathbf{I}-\mathbf{P})\mathbf{u}_n$ and $\mathbf{B}\mathbf{P}\tilde{\mathbf{u}}_m$ to alter their structure to utilize $\mathbf{R}$ and $\mathbf{F}$ to make the coupling between coarse and fine explicit.

3.1. LTS-algorithm with selection matrices

A closer look at Algorithm 1 yields the coarse region term, which can be rewritten using selection matrix $\mathbf{R}$ and its properties (33) as

$$\mathbf{w} = \mathbf{B}(\mathbf{I}-\mathbf{P})\mathbf{u} = \mathbf{P}\mathbf{B}\mathbf{R}\mathbf{u}_n + (\mathbf{I}-\mathbf{P})\mathbf{B}(\mathbf{I}-\mathbf{P})\mathbf{u}_n.$$

This explicitly gives us the contribution of the coarse nodes to fine nodes ($\mathbf{PBRu}_n$) and the coarse node intermediate step ($(\mathbf{I}-\mathbf{P})\mathbf{B}(\mathbf{I}-\mathbf{P})\mathbf{u}_n$), which only contributes to coarse nodes. Conversely, the fine-node contribution to the coarse nodes is determined from the fine-node term using selection matrix $\mathbf{F}$ properties (34),

$$\tilde{\mathbf{a}}_{m+\frac{1}{2}} = \mathbf{BP}\tilde{\mathbf{u}}_m = \mathbf{PBP}\tilde{\mathbf{u}}_m + \mathbf{RBF}\tilde{\mathbf{u}}_m.$$

Similarly, we see the fine-node contribution to other fine nodes in $\mathbf{PBP}\tilde{\mathbf{u}}_m$ and the fine-node contribution to the coarse nodes $\mathbf{RBF}\tilde{\mathbf{u}}_m$. Because the coarse-to-coarse contributions $(\mathbf{I}-\mathbf{P})\mathbf{B}(\mathbf{I}-\mathbf{P})\mathbf{u}_n$ are simply added at each substep (without mixing), these can be combined and moved from the fine region to the final, coarse-region update.

To rewrite the two-level LTS-Newmark scheme such that it can be implemented efficiently, we now introduce the reduced equality operator ($\overset{\mathbf{Q}}{=}$) that only operates on the set of nodes $\mathbf{Q}$, where $\mathbf{Q}$ represents a selection matrix. Thus, $\mathbf{y} \overset{\mathbf{Q}}{=} \mathbf{x}$ implies that only entries y_i are set to x_i where $\mathbf{Q}_{ii} = 1$. This

Algorithm 2 Two-level LTS-Newmark with selection matrices $\mathbf{R}$ and $\mathbf{F}$

Require: $\mathbf{u}_0, \mathbf{v}_{-\frac{1}{2}}, \Delta\tau = \frac{\Delta t}{p}$
  for $n = 0, 1, \ldots$ **do**
    $\mathbf{w} \overset{\mathbf{P}}{=} \mathbf{PBRu}_n$
    $\tilde{\mathbf{v}}_{\frac{1}{2}} \overset{\mathbf{P}+\mathbf{R}}{=} \frac{1}{2}\Delta\tau\left(\mathbf{w} + \mathbf{PBP}\tilde{\mathbf{u}}_0 + \mathbf{RBF}\tilde{\mathbf{u}}_0\right)$
    $\tilde{\mathbf{u}}_1 \overset{\mathbf{P}+\mathbf{R}}{=} \mathbf{u}_n + \Delta\tau\tilde{\mathbf{v}}_{\frac{1}{2}}$
    for $m = 1, \ldots, (p-1)$ **do**
      $\tilde{\mathbf{a}}_m \overset{\mathbf{P}+\mathbf{R}}{=} \mathbf{w} + \mathbf{PBP}\tilde{\mathbf{u}}_m + \mathbf{RBF}\tilde{\mathbf{u}}_m$
      $\tilde{\mathbf{v}}_{m+\frac{1}{2}} \overset{\mathbf{P}+\mathbf{R}}{=} \tilde{\mathbf{v}}_{m-\frac{1}{2}} + \Delta\tau\tilde{\mathbf{a}}_m$
      $\tilde{\mathbf{u}}_{m+1} \overset{\mathbf{P}+\mathbf{R}}{=} \tilde{\mathbf{u}}_m + \Delta\tau\tilde{\mathbf{v}}_{m+\frac{1}{2}}$
    end for
    $\mathbf{z} \overset{\mathbf{P}+\mathbf{R}}{=} 2\left(\frac{\tilde{\mathbf{u}}_p - \mathbf{u}_n}{\Delta t}\right)$
    $\mathbf{v}_{n+\frac{1}{2}} = \mathbf{v}_{n-\frac{1}{2}} + \mathbf{z} + \Delta t\,(\mathbf{I}-\mathbf{P})\mathbf{B}(\mathbf{I}-\mathbf{P})\mathbf{u}_n$
    $\mathbf{u}_{n+1} = \mathbf{u}_n + \Delta t\,\mathbf{v}_{n+\frac{1}{2}}$
  end for

rewrite of the two-level scheme is mathematically identical to the original Algorithm 1. However, by taking advantage of the reduced equality operator ($\overset{\mathbf{Q}}{=}$), operations such as $\tilde{\mathbf{v}}_{m+\frac{1}{2}} \overset{\mathbf{P}+\mathbf{R}}{=} \tilde{\mathbf{v}}_{m-\frac{1}{2}} + \Delta\tau\tilde{\mathbf{a}}_m$ and $\tilde{\mathbf{u}}_{m+1} \overset{\mathbf{P}+\mathbf{R}}{=} \tilde{\mathbf{u}}_m + \Delta\tau\tilde{\mathbf{v}}_{m+\frac{1}{2}}$ are only active on the set of fine-region nodes plus boundary nodes selected by $\mathbf{R}$ and can be implemented efficiently at a cost of $\mathrm{O}(N_{\text{fine}})$, the stated goal in Table 1. In particular, the coarse region term $(\mathbf{I}-\mathbf{P})\mathbf{B}(\mathbf{I}-\mathbf{P})\mathbf{u}_n$ has been moved from the precomputed $\mathbf{w}$ (in Algorithm 1) to $\mathbf{v}_{n+\frac{1}{2}}$ (in Algorithm 2), that is after the fine-region subproblem $\tilde{\mathbf{u}}_p = \tilde{\mathbf{u}}(\Delta t)$ has been computed. Now $\mathbf{w}$ only precomputes the necessary terms given by $\mathbf{R}$, which are used (and reused) for

$\tilde{\mathbf{v}}_{\frac{1}{2}}$ and $\tilde{\mathbf{v}}_{m+\frac{1}{2}}$. These optimizations allow for the efficient implementation of the two-level version of the scheme, but a multilevel approach will achieve even higher performance.

4. Multilevel LTS-Newmark Method

For simplicity, we previously limited ourselves to only two levels, but many applications are able to benefit significantly by adding the ability to step using an arbitrary number of levels. By allowing multiple levels, we add flexibility to the time steps, such that more elements are closer to their own optimal time step.

A move to multiple refinement levels requires further variables, which we index by level k with $k = 1, \ldots, k_{\max}$ from coarsest to finest; the following variables are defined:

1. $\mathbf{P}_k$ is a diagonal matrix with value 1 when the diagonal entry corresponds to level-k DOFs with the following properties,

$$\sum^{N} \mathbf{P}_k = \mathbf{I}, \quad \mathbf{P}_j \mathbf{P}_k = 0, j \neq k \, . \tag{35}$$

2. $\mathbf{u}_m^{(k)} = \mathbf{P}_k \mathbf{u}, \mathbf{v}_m^{(k)} = \mathbf{P}_k \mathbf{v}$, and $\mathbf{a}_m^{(k)} = \mathbf{P}_k \mathbf{a}$ at step m.
3. m is the step relative to the coarser neighbor (of $\frac{p_k}{p_{k-1}}$ steps before this level is complete).

With two levels, we simply had $\Delta\tau = \frac{\Delta t}{p}$, which gets extended to include more levels with time steps

$$\Delta\tau^{(k)} = \frac{\Delta t}{p(k)} \tag{36}$$

defined by the level k and the refinement $p(k)$. For this current method, we are restricted to refinements $p_k = p(k)$ such that each successive time step is an even divisor of all previous levels,

$$\frac{p_2}{p_1} \in \mathbb{N}, \qquad p_2 \geq p_1.$$

With even divisors, the synchronization between levels happens at every time step (except the finest level). For instance, two neighboring refinement levels can have time steps equal to $\Delta t/2$ and $\Delta t/4$, but not $\Delta t/2$ and $\Delta t/3$.

4.1. LTS-algorithm for three levels

For simplicity, we derive only a three-level version, mindful of the many-level generalization. Following the original two-level derivation, the ODE system used $\mathbf{u}(t) = \mathbf{u}^{(1)}(t)$, $\mathbf{v}(t) = \mathbf{v}^{(1)}(t)$, $\tilde{\mathbf{u}}(\tau) = \mathbf{u}^{(2)}(\tau)$, and $\tilde{\mathbf{v}}(\tau) = \mathbf{v}^{(2)}(\tau)$, however, with

the introduction of the third embedded subproblem with variables $\mathbf{u}^{(3)}(s)$ and $\mathbf{v}^{(3)}(s)$, one gets

$$\begin{aligned}\frac{d\mathbf{u}^{(2)}}{d\tau}(\tau) &= \mathbf{v}^{(2)}(\tau),\\ \frac{d\mathbf{v}^{(2)}}{d\tau}(\tau) &= \mathbf{B}\mathbf{P}_1\mathbf{u}^{(1)}(t)+\mathbf{B}\mathbf{P}_2\mathbf{u}^{(2)}(\tau)+\mathbf{B}\mathbf{P}_3\mathbf{u}^{(3)}(s).\end{aligned}$$

This third variable is governed by the following system,

$$\begin{aligned}\frac{d\mathbf{u}^{(3)}}{ds}(s) &= \mathbf{v}^{(3)}(s),\\ \frac{d\mathbf{v}^{(3)}}{ds}(s) &= \mathbf{B}\mathbf{P}_1\mathbf{u}^{(1)}(t)+\mathbf{B}\mathbf{P}_2\mathbf{u}^{(2)}(\tau)+\mathbf{B}\mathbf{P}_3\mathbf{u}^{(3)}(s),\end{aligned}$$

where $\mathbf{u}^{(1)}(t)$ and $\mathbf{u}^{(2)}(\tau)$ are constant, $\mathbf{u}^{(3)}(0) = \mathbf{u}^{(2)}(\tau)$, and $\mathbf{v}^{(3)}(0) = 0$, precisely as in the definition of $\tilde{\mathbf{u}}(\tau)$. In a recursive manner, to solve $\mathbf{u}^{(1)}(t_i+\Delta t)$, we must solve $\mathbf{u}^{(2)}(\Delta t)$, which requires $i = \Delta t/\Delta\tau_2$ steps. However, each $\Delta\tau_2$ step require us to solve $\mathbf{u}^{(3)}(\Delta\tau_2)$, requiring $j = \Delta\tau_2/\Delta\tau_3$ steps of $\Delta\tau_3$. In order to make this explicit, one can write this three-level scheme in Algorithm 3, where special care is needed at the intermediate level, when $m = 0$.

Additionally, the theoretical speedup model (28) is adapted to multiple levels,

$$\text{theoretical LTS speedup} = \frac{p_{\max}\times\#\{\text{all elements}\}}{\sum_{k=1}^{k_{\max}} p_k\times\#\{\text{elements level } k\}}. \tag{37}$$

The multilevel LTS time-stepping algorithm is best understood recursively, such that each level k is embedded within its coarser level $(k-1)$, up to the coarsest level $k = 1$. For three levels with $p_k = 1, 2, 4$, one would step with step sizes in the following order:

$$\left\{\frac{\Delta t}{4}, \frac{\Delta t}{4}, \frac{\Delta t}{2}, \frac{\Delta t}{4}, \frac{\Delta t}{4}, \frac{\Delta t}{2}, \Delta t\right\}. \tag{38}$$

Proposition 1 can be extended for this multilevel case with a multilevel equivalent $\mathbf{B}_p$ from (31) with more details available in [33].

5. Multilevel LTS-Newmark Method: Evaluation and Validation

To analyze the new multilevel LTS-Newmark method in 3D, we implemented the scheme into the community package SPECFEM3D_Cartesian [32] (referred to as SPECFEM3D from here on). This package is a comprehensive Fortran code implementing both the (visco)elastic and acoustic wave equation on large heterogeneous domains, with a focus on local and regional seismology. As a SEM, it uses hexahedral elements to enable the construction of a diagonal mass matrix, usually with $\mathcal{P}^4$ elements. Originally written in Fortran90 and using

Algorithm 3 Three-level LTS-Newmark

Require: $\mathbf{u}_0, \mathbf{v}_{-\frac{1}{2}}, \Delta\tau_2 = \frac{\Delta t}{p_2}, \Delta\tau_3 = \frac{\Delta\tau}{p_3}$

for $n = 0, \ldots, T_n$ **do**

$\mathbf{u}_0^{(2)} = \mathbf{u}_n^{(1)}$

$\mathbf{w}_1 = \mathbf{B}\mathbf{P}_1\mathbf{u}_n^{(1)}$

for $m = 1, \ldots, (p_2 - 1)$ **do**

$\mathbf{u}_0^{(3)} = \mathbf{u}_m^{(2)}$

$\mathbf{w}_2 = \mathbf{B}\mathbf{P}_2\mathbf{u}_m^{(2)}$

$\mathbf{v}_{\frac{1}{2}}^{(3)} = \frac{1}{2}\Delta\tau_3\left(\mathbf{w}_1 + \mathbf{w}_2 + \mathbf{B}\mathbf{P}_3\mathbf{u}_0^{(3)}\right)$

$\mathbf{u}_1^{(3)} = \mathbf{u}_0^{(3)} + \Delta\tau_3\mathbf{v}_{\frac{1}{2}}^{(3)}$

for $s = 1, \ldots, (p_3/p_2 - 1)$ **do**

$\mathbf{v}_{s+\frac{1}{2}}^{(3)} = \mathbf{v}_{s-\frac{1}{2}}^{(3)} + \Delta\tau_3\left(\mathbf{w}_1 + \mathbf{w}_2 + \mathbf{B}\mathbf{P}_3\mathbf{u}_s^{(3)}\right)$

$\mathbf{u}_{s+1}^{(3)} = \mathbf{u}_s^{(3)} + \Delta\tau_3\mathbf{v}_{s+\frac{1}{2}}^{(3)}$

end for

$$\mathbf{v}_{m+\frac{1}{2}}^{(2)} = \begin{cases} (\mathbf{u}_{p_3/p_2}^{(3)} - \mathbf{u}_m^{(2)})/\Delta\tau_2, & m = 0 \\ \mathbf{v}_{m-\frac{1}{2}}^{(2)} + 2(\mathbf{u}_{p_3/p_2}^{(3)} - \mathbf{u}_m^{(2)})/\Delta\tau_2, & m > 0 \end{cases}$$

$\mathbf{u}_{m+1}^{(2)} = \mathbf{u}_m^{(2)} + \Delta\tau_2\,\mathbf{v}_{m+\frac{1}{2}}^{(2)}$

end for

$\mathbf{v}_{n+\frac{1}{2}}^{(1)} = \mathbf{v}_{n-\frac{1}{2}}^{(1)} + 2(\mathbf{u}_{p_2}^{(2)} - \mathbf{u}_n^{(1)})/\Delta t$

$\mathbf{u}_{n+1}^{(1)} = \mathbf{u}_n^{(1)} + \Delta t\,\mathbf{v}_{n+\frac{1}{2}}^{(1)}$

end for

MPI for parallelization, it has been extended to work on GPU clusters using CUDA [34].

Focusing on applications in seismology, we test the implementation using localized earthquake sources (usually within a single element) and measure the resulting solutions at or near the surface at localized stations, modeling a real-world simulation scenario. Figure 3 illustrates the designed mesh and experimental setup used, with coarse boundaries and refinement in the center. For a homogeneous, isotropic elastic medium ($v_p = 2.8km/s$, $v_s = 1.5km/s$ and $\rho = 2.3kg/m^3$), we model an earthquake using a pressure-type moment-tensor solution with a Gaussian source time function (half-duration of 5 s) near the center of the mesh at 25 km depth. Two linear arrays of stations to record solutions are placed at the surface, with waves passing through all three refinement levels.

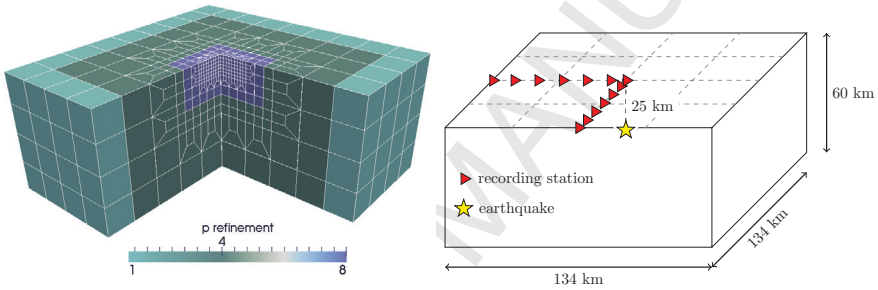


Figure 3: Cutaway of a block mesh with three local refinement levels (left) and the experimental setup (right). The smallest elements in the middle of the model require an 8x smaller time step size than the coarsest elements located at the boundary. Experimental setup has 13 surface recording points and an earthquake at 25 km depth.

Figure 4 depicts a recording of the vertical ($z-$dir) displacement of a centrally located station, with absorbing (ABC) and reflecting (non-ABC) boundaries comparing the reference solution to our new LTS-Newmark scheme. As seen by the overlap of the two solutions, the seismogram recordings match very well for both absorbing and reflecting boundary conditions. Both panels depict the arrival of the pulse from the source, where the absorbing boundaries imperfectly absorb the incident waves, allowing the wavefield to return to a near steady state, in contrast to the reflecting condition, where the waves continue to bounce around the domain. Both the LTS and reference solutions overlap perfectly, validating the implementation in SPECFEM3D for this common use case.

5.1. Performance metrics

Having demonstrated correctness in the previous section, we now demonstrate the efficiency of our solution. After all, to be practically useful, the LTS version must provide a useful speedup over the non-LTS reference version as

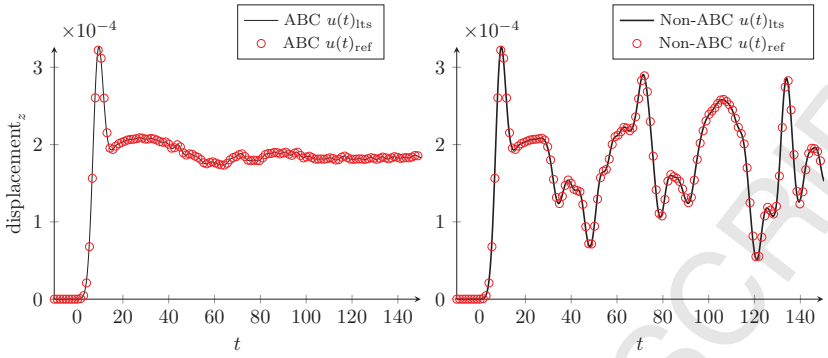


Figure 4: Seismogram comparison between LTS (black line) and non-LTS (red circles) for both absorbing (left) and nonabsorbing (right) boundaries. Both panels show the initial arrival of a wave created by the source, where the absorbing boundary solution (left) returns to a steady state with some spurious reflections due to the imperfect absorption. The non-absorbing case (right) shows the full reflections from all surfaces. Both LTS and non-LTS solutions in both panels overlap perfectly, validating this test example.

predicted by the theoretical LTS speedup model (37) for a given mesh. In order to measure the actual speedup, several performance metrics evaluate the effectiveness of the LTS version. Given that LTS is a work saving feature, we measure performance as

$$\text{Performance} = \frac{[\text{simulated time (s)}]}{[\text{elapsed time (s)}]} \tag{39}$$

where simulated time = (number of steps) $\times \Delta t$ and elapsed time measures how long a simulation took (in elapsed or computation time). Thus, for the application scientist, increased performance reduces the elapsed time required to reach a particular simulation goal. Following this measure of performance, LTS speedup is simply defined as the ratio of LTS and non-LTS performance,

$$\text{LTS speedup} = \frac{\text{Performance}_{(\text{LTS})}}{\text{Performance}_{(\text{Non-LTS})}}. \tag{40}$$

It follows then that

$$\text{LTS efficiency} = \frac{\text{actual LTS speedup}}{\text{theoretical LTS speedup}}. \tag{41}$$

For example, the validation setup from Figure 3 has a theoretical speedup of 1.3x, and our implementation in SPECFEM3D with LTS runs 1.3x faster giving it 100% LTS efficiency.

5.2. LTS efficiency in 3D

In the following, we wish to ensure that the desired overhead complexities from Table 1 are correctly implemented. Given that the mesh used to validate

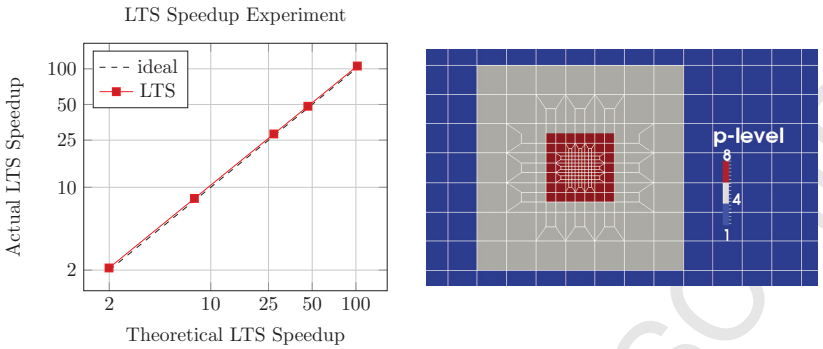


Figure 5: LTS speedup test on a 300,000 element (19M DOFs) mesh with a center refinement that is scaled from 2x–100x LTS speedup over the non-LTS version. (left) Performance results showing nearly perfect actual LTS speedup vs. theoretical LTS speedup. (right) A zoom of the refinement in the approximately 8x-speedup mesh, which has three time-stepping levels: $\Delta t, \Delta t/4, \Delta t/8$.

the 3D implementation had only a very modest LTS speedup potential, a series of meshes (1120km x 1120km x 780km) with approximately 300,000 elements has been created. The coarsest elements are 15km across. Using CUBIT's refine feature we then created elements that are 3x smaller than the original chosen element, with a surrounding transition radius. By sequentially refining the center element (and saving each resulting mesh), the element sizes ranged such that the theoretical LTS speedups (37) ranged from 2x to 100x relative to the non-LTS version of the code. To create this, the upper surface center was refined between 1 and 5 times to create time-stepping zones between $p = 2$ and $p = 256$. The final, finest refinement case creates many levels in-between $p = 1$ and $p = 256$, making it an ideal test case for the efficiency of the implementation.

Performance experiments for different meshes that compare LTS and non-LTS single-threaded execution of the code are depicted in Figure 5. One can see that the actual LTS speedup matches the theoretical LTS speedup perfectly. By ensuring that the LTS operations scale with the number of elements in each LTS level, the algorithm achieved excellent efficiency. For example on the 100x speedup mesh, the finest level has $p = 256$ and only 1,136 elements (compared to 298,000 on the coarsest level with $p = 1$). Without the analysis in Section 3, the time-stepping operations for the lowest level would be performed 256 times per global step on all 19M DOFs, heavily decreasing the performance. The near perfect efficiency shown in Figure 5 demonstrates that, for a single-threaded application, the overhead introduced by an efficient LTS implementation is minimal.

In our case, for example, SPECFEM3D uses a matrix-free implementation of the action $\mathbf{Ku}$, which takes advantage of the tensorized Lagrange basis functions and the corresponding quadrature rule to compute the element-wise matrix-vector product on the fly, trading arithmetic computation for precomputed matrix memory transfers for a performance boost on memory bandwidth

constrained algorithms and devices. Thus, the selection matrices **P**, **R**, and **F** for each refinement level must be implemented by restricting loop indexes and zeroing fields to achieve the same effect without sacrificing the performance characteristics of the original **Ku** action. The detailed description of the matrix-free implementation goes beyond the scope of this paper, and is dependent on the underlying implementation of the method. We refer interested readers to the code itself, which is being made available[5].

Part of the effectiveness seen in this 2–100x scaling example is due to choices made about the memory layout. The recursive nature of the LTS scheme ensures that the work done on finest refinement levels (which should contain a small number of elements), is repeated several times, which we try to leverage for additional performance. By organizing the degrees of freedom in memory by refinement level, the CPU version maintains a higher level of cache usage relative to the non-LTS version. In fact, this improved cache usage is amplified as more processors are added to the problem, which we discuss in the next section.

6. Multilevel LTS-Newmark Method on Massively Parallel Architectures

Having validated the 3D implementation in SPECFEM3D in single-threaded mode, we turn to much larger examples to prove the performance on synthetic and application meshes across a large number of CPU and GPU nodes. SPECFEM3D is a highly optimized code for HPC simulations, able to run on very large parallel architectures. Thus, the implementation of our new multilevel LTS-Newmark scheme has to perform very efficiently in parallel and be aware of strong memory constraints on data allocation. We begin with an outline on the parallel implementation, followed by strong-scaling experiments on a relatively large number of CPU and GPU supercomputing nodes.

6.1. Load balancing

We present here an overview of the parallelization of our multilevel LTS algorithm, providing main results to demonstrate the effectiveness of our parallelization solution for LTS on both CPU and GPU clusters for large-scale problems. A more detailed analysis of the parallelization of our LTS-Newmark method is beyond the scope of this article, but is examined and compared with several parallelization methods in a concurrent article [35].

In general, the parallelization is done by spatially partitioning the mesh, giving one partition to each MPI process. At each time-step, the shared boundary nodes between partitions are swapped between neighbors to finalize the FEM assembly. This can be done asynchronously, by computing the update to the shared partition boundaries first. The SPECFEM3D GPU version additionally overlaps CPU-GPU memory transfers with the non-boundary updates in order to hide the required communications with computation [34].

[5] `http://github.com/rietmann/specfem3d_lts`

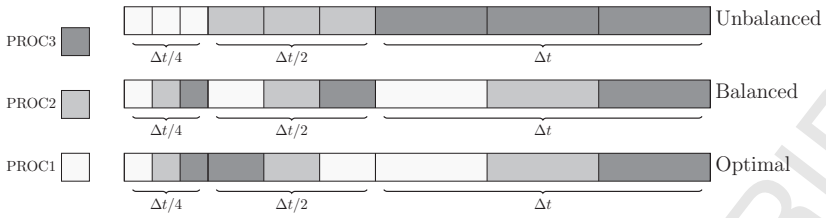


Figure 6: A 1-D mesh with nine elements and three LTS-levels partitioned across three processes that compares a worst-case unbalanced partitioning with the optimal case for our LTS load-balancing strategy. The unbalanced case would run serially because the $\Delta t/4$ level needs to complete on PROC1 before the $\Delta t/2$ level can begin on PROC2, and so on. Both the balanced and optimal case would be correctly load balanced, but the optimal case would have less communication at the $(\Delta t/4, \Delta t/2)$ and $(\Delta t/2, \Delta t)$ boundaries.

Unfortunately, an LTS scheme creates a strong load-imbalance across partitions. As noted in Section 4, the multilevel LTS algorithm can be viewed recursively, where the finest levels must complete several steps before upper, coarser levels can continue. These finer levels are also computationally more expensive, as they are completing more steps than a coarser level. Thus, the elements in each refinement level are associated with a different cost, and for a general, non-constrained partitioning, the distribution of elements in each refinement level will not be balanced across partitions (and processors). Fig. 6 demonstrates our partitioning approach, which depicts a 1-D mesh with three p-levels with time step sizes $(\Delta t/4, \Delta t/2, \Delta t)$ and the partitioning across three processes (PROC1,PROC2,PROC3) given by colors of gray.

An LTS-unaware partitioning scheme tries to balance the mesh across processors (three in this figure), while minimizing cuts between elements (and thus MPI communications). The first partitioning shown (Unbalanced) accomplishes a perfect partitioning for the non-LTS case, but represents the worst-case when LTS is active. The three-level LTS case travels through the levels recursively, as detailed in Section 4.1 and Eq. 38. In this "Unbalanced" partitioning, PROC1 must finish the finest level before PROC2 can take a step, which repeats until PROC2 finishes the intermediate level and PROC3 can take a step. This forces the three processes to run one-at-a-time, serializing the operation and eliminating any parallelism.

In contrast to the unbalanced case, we additionally depict balanced and optimal partitionings. The balanced case simply ensures that the elements in each p-level are distributed equally across all three processors, enabling the full parallelism available. However, a partitioning should also try to minimize cuts across processors to reduce necessary MPI communications, which the optimal case achieves. By grouping the $(\Delta t/4, \Delta t/2)$ and $(\Delta t/2, \Delta t)$ boundary elements on a single processor, the required amount communications is less than the balanced partitioning, which does not take the cost of cuts into account.

The dual requirements to both balance the refinement levels equally and also minimize communications can be represented as a multi-constraint partitioning problem. The SCOTCH [31] partitioner traditionally used by SPECFEM3D

however, allows only for a single constraint, making it just suitable for a two-level scheme, without further modifications. More recently, both the MeTiS [18] graph partitioning library and the PaToH [3] hypergraph partitioning library can perform multi-constraint partitioning, however, MeTiS is not currently able to adequately maintain the load-balance across levels as the parallelism is increased. PaToH, a hypergraph partitioning library, was both able to maintain the load-balance and more accurately model the more complex communications relationship between elements at different refinement levels. This approach naturally produces partitioning similar to the "Optimal" case shown in Fig. 6.

In contrast to the all-at-once approach of the multi-constraint partitioning by PaToH, we tested a simpler approach using a single-constraint partitioning scheme on each level independently. Without additional care, this approach will create the "Balanced" partitioning in Fig. 6. The assembly of partitions across refinement levels can be seen as a modified traveling salesman problem, as one tries to combine the partitions in a communication-optimal way. Fortunately, a simple greedy, first processor takes its best match, recombination approach works very well in practice for the examples tested. In fact, it remains our partitioning scheme of choice, performing as well or better than the multi-constrain approach that uses the PaToH partitioning library, however the difference was small. The examples shown in the next section's performance experiments used the simpler approach, however for more details and a comparison of the two schemes, see our concurrent paper focused entirely on the partitioning problem [35].

6.2. Parallel performance evaluation

We conducted strong-scaling benchmarks on two hybrid CPU-GPU clusters:

Piz Daint — a Cray XC30 system[6], with a single 8-core Intel XEON E5 CPU and a single K20X Kepler-generation NVIDIA GPU per node. The SPECFEM3D CPU version runs 1 process per core (8 per node) and the GPU version runs 1 process per GPU (1 per node);

Tödi — a Cray XK7 system, with a single 16-core AMD Opteron 6272 CPU and a single K20X GPU per node. The 16 CPU cores share 8 floating point modules, so the SPECFEM3D CPU version is run with 8 processes per node to match 1 process per floating point module.

To evaluate the parallel performance of our implementation, elapsed time of the LTS and non-LTS (reference) version of the code are measured for a particular mesh and desired simulation time. Given that LTS is a work saving feature, we compare the performance (defined in Section 5.1) of the LTS and non-LTS versions of SPECFEM3D.

[6] #6 supercomputer (Top500) worldwide as of June 2015.

6.2.1. Synthetic example

The mesh used to evaluate the single-threaded LTS speedup was a simple block with a variable refinement in the middle with around 300,000 elements. In order to test and optimize LTS performance at large parallelism, we extended this mesh by a factor of 30x more elements to model a possible real-world example of localized refinement. This larger mesh has 10M elements with an element-size distribution that yields a theoretical speedup of 8x.

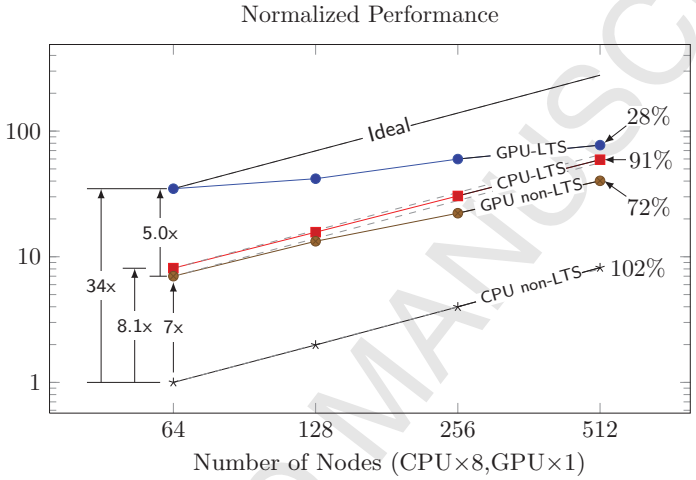


Figure 7: Synthetic CPU and GPU benchmarks on 10M element mesh with four levels of refinement $(\Delta t, \Delta t/2, \Delta t/4, \Delta t/8)$ localized to a single area at the center, with an element distribution creating 8x theoretical LTS speedup. Performance is normalized to the CPU reference (non-LTS) version at 64 nodes. The parallel scaling efficiency is listed to the right of each scaling curve.

Figure 7 presents strong-scaling experiments conducted on the *Piz Daint* cluster where both CPU-only and GPU version with multilevel LTS-Newmark and non-LTS Newmark schemes were run. Normalized to the performance of the 64-node CPU non-LTS version, the CPU-LTS version initially achieves over 100% of the theoretical speedup of 8x, and scales to 512 nodes (4096 cores) with 91% efficiency. We note that the reference version scales superlinearly, with 102% efficiency at 512 nodes. CPU profiling indicates that this is the result of better cache utilization as the partition size shrinks. The CPU-LTS version also benefits from the improved cache performance as the partition size shrinks. As noted in Section 5.2, our LTS implementation groups the degrees of freedom by refinement level. Given the recursive nature of the LTS algorithm, the inefficiencies introduced by having very few elements in the finest levels are partially offset by cache-utilization improvements as the partitions become smaller. This helps the CPU-LTS version scale efficiently by hiding the growing LTS overhead.

We also note that the GPU-LTS version starts at 5.0x speedup over the non-LTS GPU version at 64 nodes (63% LTS efficiency), which is 4.3x faster

than the CPU-LTS version (and 34x faster than the non-LTS CPU version). At higher node counts, the finest levels contain relatively few elements and are not able to keep the GPU adequately busy to mask the overhead of setting up and launching the CUDA compute kernels. This means that a strong-scaling roll off in efficiency occurs earlier than usual, which is an expected weakness of a multi-constraint partitioning approach.

6.2.2. Application example for the Tohoku-Oki subduction zone

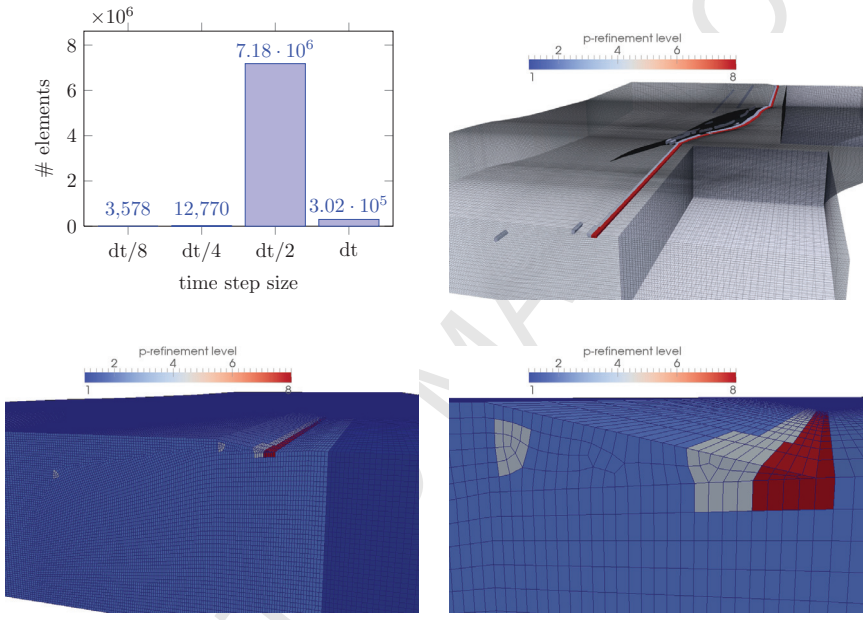


Figure 8: Tohoku mesh with 7.5M elements and a predicted speedup of 4.1 with the given element distribution.

With the excellent performance on large synthetic benchmarks, we turn to a real-world example for further performance validation. For many seismic applications, a mesh is designed to support a minimum wavelength, which is usually dependent on the resolution of the velocity structure of the medium. However, to represent internal or external structures, it is common that lines or regions of small elements occur due to meshing difficulties or the dimensions of the structure. Traditionally in SPECFEM3D and other seismic simulation codes, earthquakes are modeled as point sources, or possibly a collection of point sources. These sources simply prescribe the slip at static mesh points, and do not represent the dynamic triggering of a true fault. Seismologists are now trying to model the earthquake as a dynamic fault rupture, which requires solving dynamic rupture physics on a static mesh.

Seen in Figure 8, our application mesh is designed to model the fault slip of the subduction zone in Japan, the source of the Tohoku-Oki magnitude 9.0

earthquake in 2011 [12]. SPECFEM3D readily incorporates dynamic rupture physics and this mesh honors the internal topography of the fault surface. Note that when this internal fault reaches the surface of the mesh, some elements are squeezed and become significantly smaller (the thin dark red stripe of elements), which are 4–8 times smaller than the largest interior elements. These small elements along the fault strongly impact the efficiency of a standard explicit time-stepping scheme and have a predicted speedup of 4.1x.

Figure 9 presents strong-scaling experiments on the *Tödi* cluster (Cray XK7), where the LTS-CPU version is 3.9x faster (in runtime) as compared to the non-LTS CPU version. This corresponds to an LTS efficiency of 95%. The LTS GPU version managed an additional 7.4x speedup, yielding 28.9x total speedup against the original CPU version. The high CPU and GPU LTS efficiency indicates that our partitioning solution is working well for this real-world application example. To put these speedup numbers in perspective, using 40 nodes, the non-LTS CPU version requires more than 57 min to finish a simulation, whereas the LTS GPU version can finish in less than 2 min.

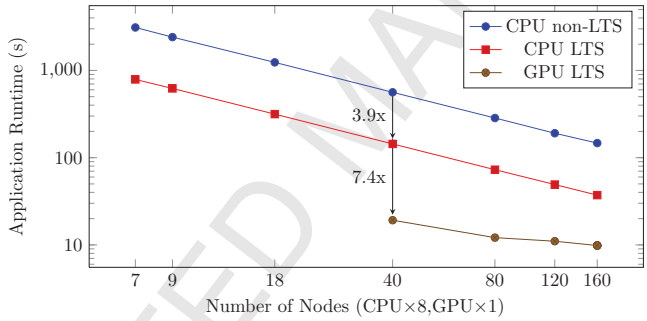


Figure 9: Runtime scaling (in seconds) comparing reference (non-LTS), LTS CPU, and LTS GPU versions running on the Tohoku mesh of 7.5M elements (and 4.1x predicted LTS speedup). The memory constraints limited the GPU version to start at 40 nodes.

7. Conclusion

We have presented a new LTS-Newmark scheme and its high-performance implementation for large-scale wave propagation simulations. The algorithm is able to utilize multiple refinement levels, yielding better performance than a simpler two-level scheme, while still maintaining the properties of the two-level version. We also provide the algorithmic improvements necessary to efficiently implement the scheme in a continuous finite-element spatial discretization such as the SEM.

To validate these improvements for practical purposes, we highlighted the implementation of our multilevel LTS-Newmark scheme in a widely-used, seismological community code, SPECFEM3D. The algorithm was integrated into both the CPU and GPU versions of the code, allowing future users to benefit

from both algorithmic and hardware optimizations. The performance experiments show that LTS-Newmark is able to fully achieve the theoretical speedup given by meshes traditionally limited by the CFL condition. The implementation is able to maintain a high LTS efficiency even on meshes with a factor of 100x theoretical speedup. Given that this can be combined with GPU speedup, the new version of the code is possibly an order of magnitude faster than the original CPU reference code. Of course, this speedup depends on the mesh in question, which is generally designed for a particular application or experiment.

We demonstrate the LTS speedup on larger-scale seismic synthetic and application examples, with simulations run on a relatively large number of CPUs and GPUs. Through a multi-constraint partitioning approach, the multilevel LTS implementation gets effectively load-balanced across hundreds of multicore CPU and GPU compute nodes. As expected, strong-scaling for the LTS-GPU version suffers for very small element counts in one of the refinement levels, whereas as the LTS-CPU version still remains efficient. For more modest parallelism goals, combining LTS with GPU computations still remains very effective, providing a speedup of nearly 30x over the non-LTS CPU reference version for a practical example highlighted in Section 6.2.2. Thus, for meshes with localized small elements creating a strong CFL bottleneck, LTS-Newmark provides an effective algorithmic solution that can be implemented for high-performance computing architectures.

Finally, we want to emphasize that the multilevel LTS scheme developed here is not tied to seismic wave propagation simulations, but can certainly be applied to acoustic and electromagnetic problems as well. It would also be interesting to further investigate the stability behaviour for absorbing boundaries conditions in contact with fine-region elements in future work, as mentioned in Section 2.3. Furthermore, combining the LTS scheme with different orders of interpolation in space and time would be attractive to evaluate, especially within a discontinuous Galerkin method approach [1]. Although higher-order conservative time schemes exist, one could investigate how to increase the order of Newmark schemes as employed here for mixing both spatial and temporal orders in future LTS studies.

Acknowledgments

The computational resources and services used in this work were provided by the Swiss National Supercomputing Centre (CSCS). D. Peter and M. Rietmann were supported by the Swiss PASC project "A framework for multi-scale seismic modelling and inversion." SPECFEM3D_Cartesian is hosted by the Computational Infrastructure for Geodynamics (CIG) which is supported by the National Science Foundation award NSF-0949446.

[1] C. Baldassari, H. Barucq, H. Calandra, and J. Diaz, *Numerical performances of a hybrid local-time stepping strategy applied to the reverse time migration*, Geophysical Prospecting, 59 (2011), pp. 907–919.

[2] L. Carrington, D. Komatitsch, M. Laurenzano, M. Tikir, D. Michéa, N. L. Goff, A. Snavely, and J. Tromp, *High-frequency simulations of global seismic wave propagation using SPECFEM3D GLOBE on 62K processors*, in Proceedings of the Supercomputing 2008 Conference, 2008.

[3] Ü. V. Çatalyürek and C. Aykanat, *PaToH: A Multilevel Hypergraph Partitioning Tool, Version 3.0*, Bilkent University, Department of Computer Engineering, Ankara, 06533 Turkey. PaToH is available at `http://bmi.osu.edu/~umit/software.htm`, 1999.

[4] R. Clayton and B. Engquist, *Absorbing boundary conditions for acoustic and elastic wave equations*, Bulletin of the Seismological Society of America, 67 (1977), pp. 1529–1540.

[5] Computational Infrastructure for Geodynamics (CIG), *SPECFEM3D Cartesian*. `https://geodynamics.org/cig/software/specfem3d`, version 2.1, January 2015.

[6] F. A. Dahlen and J. Tromp, *Theoretical Global Seismology*, Princeton University Press, 1998.

[7] J. Diaz and M. J. Grote, *Energy Conserving Explicit Local Time Stepping for Second-Order Wave Equations*, SIAM J. Sci. Comput., 31 (2009), pp. 1985–2014.

[8] J. Diaz and M. J. Grote, *Multi-level explicit local time-stepping methods for second-order wave equations*, Computer Methods in Applied Mechanics and Engineering, 291 (2015), pp. 240–265.

[9] M. Dumbser, M. Käser, and E. F. Toro, *An arbitrary high-order Discontinuous Galerkin method for elastic waves on unstructured meshes - V. Local time stepping and p -adaptivity*, Geophys. J. Internat., 171 (2007), pp. 695–717.

[10] A. Fichtner, B. L. N. Kennett, H. Igel, and H.-P. Bunge, *Full seismic waveform tomography for upper-mantle structure in the Australasian region using adjoint methods*, Geophys. J. Internat., 179 (2009), pp. 1703–1725.

[11] A. Fichtner, E. Saygin, T. Taymaz, P. Cupillard, Y. Capdeville, and J. Trampert, *The deep structure of the north anatolian fault zone*, Earth and Planetary Science Letters, 373 (2013), pp. 109–117.

[12] P. Galvez, J.-P. Ampuero, L. Dalguer, S. Somala, and T. Nissen-Meyer, *Dynamic earthquake rupture modelled with an unstructured 3-d spectral element method applied to the 2011 m9 tohoku earthquake*, Geophysical Journal International, 198 (2014), pp. 1222–1240.

[13] N. Gödel, S. Schomann, T. Warburton, and M. Clemens, *GPU accelerated Adams–Bashforth multirate discontinuous Galerkin FEM simulation of high-frequency electromagnetic fields*, IEEE Trans. Magnetics, 46 (2010), pp. 2735–2738.

[14] M. J. Grote, M. Mehlin, and T. Mitkova, *Runge-Kutta Based Explicit Local Time-Stepping Methods for Wave Propagation*, SIAM J. Sci. Comput., 37 (2015), pp. A747–A775.

[15] M. J. Grote and T. Mitkova, *High-order explicit local time-stepping methods for damped wave equations*, J. Comput. Appl. Math., 239 (2013), pp. 270–289.

[16] J. Hesthaven and T. Warburton, *Nodal discontinuous Galerkin methods: algorithms, analysis, and applications*, Springer-Verlag New York Inc, Jan. 2008.

[17] T. J. R. Hughes, *The finite element method, linear static and dynamic finite element analysis*, Prentice-Hall International, Englewood Cliffs, New Jersey, USA, 1987.

[18] G. Karypis and V. Kumar, *A fast and high quality multilevel scheme for partitioning irregular graphs*, SIAM Journal on scientific Computing, 20 (1998), pp. 359–392.

[19] D. Komatitsch and J. Tromp, *Introduction to the spectral element method for three-dimensional seismic wave propagation*, Geophys. J. Internat., 139 (1999), pp. 806–822.

[20] ——, *Spectral-element simulations of global seismic wave propagation-I. Validation*, Geophys. J. Internat., 149 (2002), pp. 390–412.

[21] D. Komatitsch, S. Tsuboi, J. Chen, and J. Tromp, *A 14.6 billion degrees of freedom, 5 teraflop, 2.5 terabyte earthquake simulation on the Earth Simulator*, Proceedings of the ACM/IEEE Supercomputing SC'2003 conference, (2003). on CD-ROM.

[22] D. Komatitsch, S. Tsuboi, and J. Tromp, *The spectral-element method in seismology*, in Seismic Earth: Array Analysis of Broadband Seismograms, A. Levander and G. Nolet, eds., vol. 157 of Geophysical Monograph, American Geophysical Union, Washington DC, USA, 2005, pp. 205–228.

[23] D. Komatitsch and J. P. Vilotte, *The spectral-element method: an efficient tool to simulate the seismic response of 2D and 3D geological structures*, Bulletin of the Seismological Society of America, 88 (1998), pp. 368–392.

[24] S. Krenk, *Energy conservation in Newmark based time integration algorithms*, Comput. Methods Appl. Mech. Engrg., 195 (2006), pp. 6110–6124.

[25] T. Lay and E. Garnero, *Deep mantle seismic modeling and imaging*, Ann. Rev. Earth Planet. Sci., 39 (2011), pp. 91–123.

[26] F. Lörcher, G. Gassner, and C.-D. Munz, *A discontinuous galerkin scheme based on a space–time expansion. i. inviscid compressible flow in one space dimension*, Journal of Scientific Computing, 32 (2007), pp. 175–199.

[27] Y. Maday and A. T. Patera, *Spectral-element methods for the incompressible Navier-Stokes equations*, in State of the art survey in computational mechanics, 1989, pp. 71–143. A. K. Noor and J. T. Oden editors.

[28] S. Minisini, E. Zhebel, A. Kononov, and W. A. Mulder, *Local time stepping with the discontinuous Galerkin method for wave propagation in 3D heterogeneous media*, Geophysics, (2013).

[29] T. Nissen-Meyer, A. Fournier, and F. A. Dahlen, *A 2-D spectral-element method for computing spherical-earth seismograms–II. Background models*, Geophys. J. Internat., 174 (2008), pp. 873–888.

[30] A. T. Patera, *A spectral element method for fluid dynamics: laminar flow in a channel expansion*, J. Comp. Phys., 54 (1984), pp. 468–488.

[31] F. Pellegrini, *SCOTCH 5.1 User's Guide*, Laboratoire Bordelais de Recherche en Informatique (LaBRI), 2008.

[32] D. Peter, D. Komatitsch, Y. Luo, R. Martin, N. Le Goff, E. Casarotti, P. Le Loher, F. Magnoni, Q. Liu, C. Blitz, T. Nissen-Meyer, P. Basini, and J. Tromp, *Forward and adjoint simulations of seismic wave propagation on fully unstructured hexahedral meshes*, Geophys. J. Internat., 186 (2011), pp. 721–739.

[33] M. Rietmann, *Local Time Stepping on High Performance Computing Architectures*, PhD thesis, Universitá della Svizzera italiana, 2015.

[34] M. Rietmann, P. Messmer, T. Nissen-Meyer, D. Peter, P. Basini, D. Komatitsch, O. Schenk, J. Tromp, L. Boschi, and D. Giardini, *Forward and adjoint simulations of seismic wave propagation on emerging large-scale GPU architectures*, Proc. of the Internat. Conf. on High Perf. Comput., Netw., Stor. and Anal., (2012), p. 38.

[35] M. Rietmann, B. Ucar, D. Peter, O. Schenk, and M. Grote, *Load-balanced local time stepping for large-scale wave propagation*, in Parallel Distributed Processing Symposium (IPDPS), 2015 IEEE International, May 2015.

[36] B. Romanowicz and A. Dziewoński, *Treatise on Geophysics: Seismology and Structure of the Earth*, Elsevier, 2010.

[37] R. Snieder and E. Larose, *Extracting earth's elastic wave response from noise measurements*, Ann. Rev. Earth Planet. Sci., 41 (2013), pp. 183–206.

[38] C. Tape, Q. Liu, A. Maggi, and J. Tromp, *Adjoint tomography of the southern California crust.*, Science (New York, N.Y.), 325 (2009), pp. 988–92.

[39] A. Taube, M. Dumbser, C.-D. Munz, and R. Schneider, *A high-order discontinuous Galerkin method with time-accurate local time stepping for the Maxwell equations*, International Journal of Numerical Modelling: Electronic Networks, Devices and Fields, 22 (2009), pp. 77–103.

[40] J. Tromp, D. Komatitsch, V. Hjoerleifsdottir, Q. Liu, H. Zhu, D. Peter, E. Bozdağ, D. McRitchie, P. Friberg, C. Trabant, and A. Hutko, *Near real-time simulations of global CMT earthquakes*, Geophys. J. Internat., 183 (2010), pp. 381–389.

[41] J. Virieux and S. Operto, *An overview of full-waveform inversion in exploration geophysics*, Geophysics, 74 (2009), pp. WCC1–WCC26.

[42] H. Zhu, E. Bozdağ, D. Peter, and J. Tromp, *Structure of the European upper mantle revealed by adjoint tomography*, Nature Geoscience, 5 (2012), pp. 493–498.

Appendix A. Proof of Lemma 2.1

The proof is by induction over j. As an intermediate step, we show that $\tilde{\mathbf{v}}_{j-\frac{1}{2}}$ satisfies

$$\tilde{\mathbf{v}}_{j-\frac{1}{2}} = \sum_{i=0}^{j-1} \beta_i^j (\Delta\tau)^{2i+1} (\mathbf{BP})^i \mathbf{B}\mathbf{u}_n, \quad j \geq 1, \tag{A.1}$$

where the constants β_i^j are also recursively defined. For $j = 1$, we immediately obtain from Algorithm 1

$$\tilde{\mathbf{v}}_{\frac{1}{2}} = \frac{1}{2}\Delta\tau(\mathbf{BP}\mathbf{u}_n + \mathbf{w}) = \frac{1}{2}\Delta\tau\mathbf{B}\mathbf{u}_n,$$

which corresponds to (A.1) with $\beta_0^1 = \frac{1}{2}$. Next, for $j = 2$, we have from Algorithm 1 for $\tilde{\mathbf{v}}_{j-\frac{1}{2}}$

$$\begin{aligned} \tilde{\mathbf{v}}_{3/2} &= \tfrac{1}{2}\Delta\tau\mathbf{B}\mathbf{u}_n + \Delta\tau\mathbf{B}\mathbf{P}\tilde{\mathbf{u}}_1 + \Delta\tau\mathbf{B}(\mathbf{I}-\mathbf{P})\mathbf{u}_n \\ &= \tfrac{3}{2}\Delta\tau\mathbf{B}\mathbf{u}_n + \tfrac{1}{2}\Delta\tau^3\mathbf{B}\mathbf{P}\mathbf{B}\mathbf{u}_n, \end{aligned}$$

which corresponds to (A.1) with $\beta_0^2 = \frac{3}{2}, \beta_1^2 = \frac{1}{2}$. Using that result below, we have

$$\begin{aligned} \tilde{\mathbf{u}}_2 = \tilde{\mathbf{u}}_1 + \Delta\tau\tilde{\mathbf{v}}_{3/2} &= \tilde{\mathbf{u}}_1 + \Delta\tau(\tfrac{3\Delta\tau}{2}\mathbf{B}\mathbf{u}_n + \tfrac{\Delta\tau^3}{2}\mathbf{B}\mathbf{P}\mathbf{B}\mathbf{u}_n) \\ &= \mathbf{u}_n + \tfrac{4\Delta\tau^2}{2}\mathbf{B}\mathbf{u}_n + \tfrac{\Delta\tau^4}{2}\mathbf{B}\mathbf{P}\mathbf{B}\mathbf{u}_n, \end{aligned} \tag{A.2}$$

which corresponds to (A.1) with $\alpha_0^2 = 2$ and $\alpha_1^2 = \frac{1}{2}$.

Now, let (29) – (30) and (A.1) hold for $j-1$. Then,

$$\begin{aligned}\tilde{\mathbf{v}}_{j-\frac{1}{2}} &= \tilde{\mathbf{v}}_{j-3/2} + \Delta\tau\mathbf{w} + \Delta\tau\mathbf{B}\mathbf{P}\tilde{\mathbf{u}}_{j-1} \\ &= \sum_{i=0}^{j-2} \beta_i^{j-1} (\Delta\tau)^{2i+1} (\mathbf{B}\mathbf{P})^i \mathbf{B}\mathbf{u}_n + \Delta\tau\mathbf{B}(\mathbf{I}-\mathbf{P})\mathbf{u}_n \\ &\quad + \Delta\tau\mathbf{B}\mathbf{P}\left(\mathbf{u}_n + \sum_{i=0}^{j-2} \alpha_i^{j-1} (\Delta\tau)^{2i+2} (\mathbf{B}\mathbf{P})^i \mathbf{B}\mathbf{u}_n\right).\end{aligned}$$

We include the term $\Delta\tau\,\mathbf{B}\mathbf{u}_n$ into the second sum, shift the sum index by one, and combine the first and resulting second sum to get

$$\begin{aligned}\tilde{\mathbf{v}}_{j-\frac{1}{2}} &= (\beta_0^{j-1}+1)\Delta\tau\mathbf{B}\mathbf{u}_n + \sum_{i=1}^{j-2} (\alpha_{i-1}^{j-1} + \beta_i^{j-1})(\Delta\tau)^{2i+1}(\mathbf{B}\mathbf{P})^i\mathbf{B}\mathbf{u}_n \\ &\quad + \alpha_{j-2}^{j-1}(\Delta\tau)^{2(j-1)+2}(\mathbf{B}\mathbf{P})^{(j-1)}\mathbf{B}\mathbf{u}_n,\end{aligned}$$

which corresponds to (A.1) if

$$\begin{cases} \beta_0^j = \beta_0^{j-1} + 1, \\ \beta_i^j = \beta_i^{j-1} + \alpha_{i-1}^{j-1}, \quad 1 \le i \le j-2, \\ \beta_{j-1}^j = \alpha_{j-2}^{j-1}. \end{cases} \tag{A.3}$$

Again using Algorithm 1 and the induction hypothesis, we have

$$\begin{aligned}\tilde{\mathbf{u}}_j = \tilde{\mathbf{u}}_{j-1} + \Delta\tau\tilde{\mathbf{v}}_{j-\frac{1}{2}} &= \mathbf{u}_n + \sum_{i=0}^{j-2} \alpha_i^{j-1}(\Delta\tau)^{2i+2}(\mathbf{B}\mathbf{P})^i\mathbf{B}\mathbf{u}_n \\ &\quad + \Delta\tau\sum_{i=0}^{j-1}\beta_i^j(\Delta\tau)^{2i+1}(\mathbf{B}\mathbf{P})^i\mathbf{B}\mathbf{u}_n.\end{aligned}$$

After combining the two sums, we obtain

$$\begin{aligned}\tilde{\mathbf{u}}_j &= \mathbf{u}_n + \sum_{i=1}^{j-2}(\alpha_i^{j-1} + \beta_i^j)(\Delta\tau)^{2i+2}(\mathbf{B}\mathbf{P})^i\mathbf{B}\mathbf{u}_n \\ &\quad + \beta_{j-1}^j(\Delta\tau)^{(2(j-1)+2)}(\mathbf{B}\mathbf{P})^{(j-1)}\mathbf{B}\mathbf{u}_n,\end{aligned}$$

which corresponds to (29) if α_i^j satisfies the recursion

$$\begin{cases} \alpha_i^j = \alpha_i^{j-1} + \beta_i^j, \quad 0 \le i \le j-2, \\ \alpha_{j-1}^j = \beta_{j-1}^j. \end{cases} \tag{A.4}$$

From (A.3), we replace $\beta_0^j = \beta_0^{j-1} + 1$ and $\beta_i^j = \beta_i^{j-1} + \alpha_{i-1}^{j-1}$ in the above to yield

$$\begin{cases} \alpha_0^j = \alpha_0^{j-1} + \beta_0^{j-1} + 1, \\ \alpha_i^j = \alpha_i^{j-1} + \beta_i^j = \alpha_i^{j-1} + \beta_i^{j-1} + \alpha_{i-1}^{j-1}, \quad 1 \le i \le j-2, \\ \alpha_{j-1}^j = \beta_{j-1}^j. \end{cases}$$

From (A.4), we know that $\beta_i^{j-1} = \alpha_i^{j-1} - \alpha_i^{j-2}$, but only for $i \le j-3$. For $i = j-2$, we instead use $\alpha_{j-2}^{j-1} = \beta_{j-2}^{j-1}$. Clearly, we have $\beta_0^j = j - 1/2$ leaving us with

$$\begin{cases} \alpha_0^j = \alpha_0^{j-1} + j - \frac{1}{2}, \\ \alpha_i^j = 2\alpha_i^{j-1} - \alpha_i^{j-2} + \alpha_{i-1}^{j-1}, \quad 1 \le i \le j-2, \\ \alpha_{j-2}^j = 2\alpha_{j-2}^{j-1} + \alpha_{j-3}^{j-1}, \\ \alpha_{j-1}^j = \beta_{j-1}^j = \alpha_{j-2}^{j-1}. \end{cases}$$

Finally, it is trivial to show that $\alpha_0^j = \frac{j^2}{2}$, which completes the proof of (29) – (30). □