Technische Universität München
Fakultät für Informatik
Lehrstuhl für Wissenschaftliches Rechnen

Flexible model extension and optimisation for earthquake simulations at extreme scales

Carsten Uphoff

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender:	Prof. Dr. Harald Räcke
Prüfer der Dissertation:	1. Prof. Dr. Michael Bader
	2. Prof. Dr. David Ham

Die Dissertation wurde am 23.12.2019 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 05.03.2020 angenommen.

Acknowledgements

I like to thank my advisor, Michael Bader, for his support, for mentoring, for giving me the chance to work on this project, and for teaching me the subtleties of typography.

My gratitude goes to the whole SC '17 team, that is, Sebastian, Stephanie, Thomas, Betsy, Michael, and Alice. It is always surprising how fast things get done when everyone is highly motivated. I especially want to thank Sebastian with whom I fruitfully spent much time developing SeisSol.

My office mate Leo endured listening to my ideas and was available for discussion at all times. Thank you!

I am grateful to Alex and my new colleagues Lukas and Sebastian for proofreading the manuscript.

My special thanks goes to Dorothea for supporting and enduring me, especially in the last year, and for ensuring a work-life balance what might have otherwise been a work-work balance.

Abstract

Simulations are an indispensable tool to enhance our understanding of earthquakes. But, the modelling of realistic earthquake scenarios is challenging, and requires scalable and efficient software in combination with extensive supercomputing resources.

The discontinuous Galerkin (DG) method is a promising tool for earthquake simulation: It is flexible enough to solve a variety of scenarios, including complex rheological models of the Earth as well as dynamic simulations of the rupture process. It has been shown that a highly optimised implementation may scale to more than one hundred thousand cores with very high efficiency. However, having a sustainable and highly efficient implementation for a variety of models is a challenging software engineering problem. The problem is further complicated by new supercomputers with novel architectures that emerge every year, which require a continuous adaption of optimisations.

In this work, I introduce an abstraction layer for small tensor operations, which make up the major part of the computational kernels in the DG-code SeisSol. A domain-specific language (DSL) based on the Einstein notation is introduced, as well as a compiler which maps tensor operations to specialised code generators or to basic linear algebra subprograms. It is shown that this approach is able to reproduce the existing high performance for elastic rheological models, and that a high performance is achieved for newly implemented rheological models and dynamic rupture kernels. The DSL is also suitable to implement ensemble simulations, which might be vital to exploit future throughput-oriented architectures.

In addition, SeisSol's main data structure is generalised to support multiple rheological models, model initialisation is simplified, and local time-stepping is generalised to support dynamic rupture. The capability of the work presented in this thesis is demonstrated with several large-scale simulations, including the first dynamic rupture simulation of the 2004 Sumatra earthquake, which is to date the largest and longest of its kind.

Contents

CHAPTER

1

Introduction

Earthquakes are devastating natural events, which cause tens of thousands of casualties and damage property in the range of billions of dollars. Particularly disastrous were the 2004 Sumatra-Andaman earthquake and Indian Ocean tsunami, which is estimated to have killed 230,000 people [137], and the 2011 Tōhoku Earthquake and tsunami which led to the meltdown of the Fukushima Daiichi power plant. These earthquakes released enormous energy as seismic waves of the order of 10^{18} J. But also smaller events may cause severe damage. E.g. the 2011 Christchurch earthquake, with a moment magnitude of M_W 6.3 or about 10^{14} J, killed 184 people and caused cost in the range of billions of dollars [73].

Understanding earthquakes is challenging, as a particular outcome may be caused by the interaction of many factors, for example topography (or bathymetry), complex rheological models, or non-linear source dynamics on natural fault systems. As such, simulations are an indispensable tool in the study of earthquakes, as complex earthquake models may only be solved using numerical methods.

In the last two decades, the increasing performance of supercomputers enabled simulation software to raise the bar in the accurate resolution of high frequency seismic waves: In 2003, Komatitsch et al. [87] presented a simulation which resolves 0.2 Hz waves in global seismology, using a Spectral Element Method with $14.6 \cdot 10^9$ degrees of freedom (DOFs). Since then, the number of DOFs within a single simulation has steadily in-

creased [67, 70, 131, 134], up to $23.4 \cdot 10^{12}$ DOFs and frequencies up to 18 Hz [53].

Clearly, increasing the number of DOFs must not be an end in itself. In particular, with increasing frequency seismic waves become more sensitive to topography and small-scale material heterogeneities [72]. The amplitudes of seismic waves decrease more quickly with increasing frequency due to intrinsic attenuation, and hence a viscoelastic rheological model should be preferred to a purely elastic rheological model [134]. Moreover, non-linear deformation needs to be accounted for in the vicinity of faults [133, 164].

A zoo of numerical methods is employed in computational seismology, e.g. finite difference methods [34, 42, 53, 90, 91, 134], finite volume methods [11, 40], or continuous finite element methods [8, 37, 70, 87, 88, 131] and discontinuous finite element methods [39, 49, 67, 98, 148, 150, 162]. In this thesis, the focus lies on the discontinuous Galerkin finite element method (DG-FEM), for the following reasons: First of all, the geometric flexibility of the DG method allows to model topography, material layers, and natural fault systems. In particular with unstructured tetrahedral meshes, automatic mesh generation may be employed. Second, high-order accuracy schemes are possible [39, 68]. Third, the DG method is local, even for high-order schemes, such that it requires low communication volume and scales very well to hundreds of thousands of cores [20, 67, 158, 162].

The desired flexibility and complexity in earthquake models, in combination with the high efficiency requirements, is quite a challenge for a simulation code, which becomes evident when considering the shear number of different codes all solving the same set of equations. The DG-code SeisSol, which is the main subject of study in this thesis, is a good example: In the last two decades, many advanced earthquake models have been implemented in SeisSol and its predecessors, including the isotropic elastic wave equation [39], viscoelastic attenuation [81], anisotropy [126], poroelasticity [125], coupled elastic-acoustic media [80], off-fault plasticity [164], kinematic [83] and dynamic [123, 124] earthquake rupture models. Moreover, SeisSol has been tuned for several recent CPU architectures using code generation, giving speed-ups of 6× to 29× in comparison to the classic Fortran version [67]. The tuned versions of SeisSol [19, 22, 64, 67], however, lacked many advanced features and were restricted to isotropic elastic wave propagation with kinematic and dynamic rupture models. There has never been a version of SeisSol which would combine all features and optimisations.

A classic strategy to handle software complexity is the separation of concerns: The language used to implement the DG method should be independent of the particular hardware architecture, such that dealing with details of the numerical scheme becomes independent of hardware subtleties. In the DG scheme of SeisSol [39], and in other DG schemes [49, 98, 162], the element-update schemes consist mostly of small tensor contractions, hence those are a natural candidate for abstraction. In this thesis, a domain-specific language (DSL) is introduced for small tensor contractions. The language is based on the Einstein convention, which states that indices appearing twice are implicitly summed over. The advantage of such a language is that a DG scheme may be derived using the Einstein notation [39], and if done so, the DSL closely resembles the mathematical formulation of the numerical scheme. Furthermore, a compiler is introduced, which maps the small tensor contractions to tailored General Matrix-Matrix Multiplication (GEMM) routines [19, 66]. It is investigated whether the DSL and its compiler are able to reproduce the excellent performance of the tuned versions of SeisSol [22, 64, 67], and it is discussed whether the DSL simplifies the implementation and optimisation of advanced models, such as viscoelastic attenuation and dynamic rupture. Finally, the DSL is used to implement ensemble simulations [20], which requires invasive changes of the underlying data structures.

The DSL for small tensor contractions targets the single-node performance of a code, which is naturally limited by the maximum performance of a node. Thus, further benefits in time-to-solution also require algorithmic changes: In SeisSol, the explicit Arbitrary high-order DERivatives (ADER) time-stepping scheme allows each element to have its own time-step [41]. An efficient implementation of local-time stepping (LTS), which clusters elements with similar time-steps, was shown to deliver a speed-up of $4\times$ on 3072 nodes for a scenario with a theoretical maximum speed-up of $6.4\times$ in comparison to global time-stepping [18].

The original LTS implementation [18], on which this work is based on, is limited to kinematic source models, but in particular for dynamic rupture source models LTS may bring a huge reduction in time-to-solution. LTS is beneficial here, because the inherent length-scale in earthquake source dynamics lies in the range of metres [34], whereas seismic waves generated by earthquakes propagate over thousands of kilometres. Moreover, tiny elements might be introduced by the mesh generator at the intersection of a fault with material layers or topography which deteriorate the time-step size. Consequently, in this thesis the clustered local-time stepping scheme is extended to support dynamic rupture.

This thesis is structured as follows: Chapter 2 contains a short introduction to the physics of seismic wave propagation and earthquake source models. The employed ADER-DG scheme is briefly explained in Chapter 3. In Chapter 4, the numerical flux is discussed in detail, which is an important part of the DG scheme. Chapter 4 is an essential preliminary for Chapter 5 in which a stability analysis for the semi-discrete form of SeisSol's DG scheme is contributed, which includes dynamic rupture.

Beginning with Chapter 6, the focus lies on managing software complexity and optimisation of advanced models. The methodological foundation is laid in Chapter 6 where the DSL for small tensor contractions is presented. In Chapter 7 the DSL is applied concretely to advanced models for earthquake simulation and ensemble simulations. The extension of LTS to dynamic rupture is presented in Chapter 8. The 9th chapter stands on its own and tackles the question of rapid model initialisation.

Evaluation of this work starts with verification exercises in Chapter 10. We measure single node performance and multi-node performance in Chapter 11. In Chapter 12 we present the to date largest and longest dynamic rupture simulation of the 2004 Sumatra-Andaman earthquake. Moreover, we evaluate the capability of this work by large-scale simulations with viscoelastic attenuation.

CHAPTER

2

Earthquake physics

In the study of earthquakes, one requires a seismic wave propagation model and a description of the earthquake's source.

We briefly introduce the general ideas of the wave propagation model following the Lagrangian description of Aki et al. [3]: The deformation of a body is described by a displacement field $\boldsymbol{U}(\boldsymbol{x}, t)$, that is, a particle at $\boldsymbol{x}$ is located at $\boldsymbol{x} + \boldsymbol{U}(\boldsymbol{x}, t)$ at time t. The body's distortion is analysed by measuring the change in an infinitesimal small line element $\delta\boldsymbol{x}$ due to the displacement field. Let $\boldsymbol{x} + \delta\boldsymbol{x}$ be a point close to $\boldsymbol{x}$, the distorted line element is $\delta\boldsymbol{x} + \delta\boldsymbol{U}$ with $\delta\boldsymbol{U} = \boldsymbol{U}(\boldsymbol{x} + \delta\boldsymbol{x}, t) - \boldsymbol{U}(\boldsymbol{x}, t)$, see Figure 1.

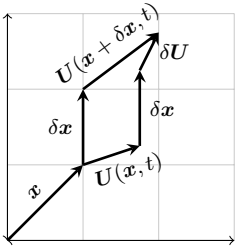


Figure 1: Sketch illustrating the distortion of a line element $\delta\boldsymbol{x}$ due to a displacement field $\boldsymbol{U}$.

From the Taylor expansion $\boldsymbol{U}(\boldsymbol{x}+\delta\boldsymbol{x},t) \approx \boldsymbol{U}(\boldsymbol{x},t) + J_{\boldsymbol{U}}(\boldsymbol{x},t)\delta\boldsymbol{x}$ we obtain the approximation $\delta\boldsymbol{U} \approx J_{\boldsymbol{U}}(\boldsymbol{x},t)\delta\boldsymbol{x}$. Thus, the Jacobian $J_{\boldsymbol{U}}$ is a measure of distortion caused by the displacement field. However, a rigid body rotation of a body would lead to a non-zero Jacobian but not to the distortion of the body. In the case that $|\frac{\partial U_i}{\partial x_j}| \ll 1$, the antisymmetric part of the Jacobian may be interpreted as rigid body rotation [3], which leaves the symmetric part of the Jacobian as measure of distortion. The latter is called *strain tensor* and is given by

$$\epsilon := \frac{1}{2}(J_{\boldsymbol{U}} + J_{\boldsymbol{U}}^T) \quad \text{or equivalently} \quad \epsilon_{ij} = \frac{1}{2}\left(\frac{\partial}{\partial x_j}U_i + \frac{\partial}{\partial x_i}U_j\right). \tag{2.1}$$

The second ingredient is Newton's second law ($\boldsymbol{F} = m\boldsymbol{a}$), which relates the displacement field to forces acting on a control volume V. The change in impulse must be equal to the force acting on the volume, which is composed of body force f_i and surface traction T_i [3]:

$$\frac{\partial}{\partial t}\int_V \rho\frac{\partial}{\partial t}U_i \,\mathrm{d}V = \int_V f_i \,\mathrm{d}V + \int_{\partial V} T_i \,\mathrm{d}S, \tag{2.2}$$

where ρ is the material density. The traction always acts on a plane with normal $\boldsymbol{n}$ and it can be shown that the traction is a linear combination of the traction acting on the three coordinate planes with normals $\boldsymbol{e}_1, \boldsymbol{e}_2, \boldsymbol{e}_3$ [3, 145]. That is, $T_i(\boldsymbol{n}) = T_i(\boldsymbol{e}_j)n_j$.[1] In the latter matrix-vector product, the matrix is called the *stress tensor* and is denoted with σ, i.e.

$$\sigma_{ji} := T_i(\boldsymbol{e}_j). \tag{2.3}$$

By inserting the traction into (2.2) and using the divergence theorem one obtains the differential form of (2.2):

$$\rho\frac{\partial^2}{\partial t^2}U_i = f_i + \frac{\partial}{\partial x_j}\sigma_{ji}. \tag{2.4}$$

Furthermore, the stress tensor is symmetric [3, 145], such that one may replace σ_{ji} with σ_{ij} in (2.4).

The missing link between (2.1) and (2.4) is the rheological model, which relates the stress tensor and the strain tensor. We introduce the rheological models used in this work in Sections 2.1 to 2.3.

[1] Summation over repeated indices is implied here and throughout this thesis.

A frequent cause of earthquakes is spontaneous rupture on a geological fault. Mathematically, this is modelled as a displacement discontinuity on an internal surface inside a body. One has to distinguish between a kinematic rupture description and a dynamic rupture description. In the former, one models the discontinuity by equivalent body forces and plugs these into (2.4). The progression of the rupture is orchestrated and does not interact with the seismic waves. In a dynamic rupture description, the internal surface is kept and a friction law is imposed as a boundary condition. The friction law typically depends on the stress tensor and particle velocity discontinuity, such that it interacts with seismic waves that pass the fault surface. We briefly introduce kinematic and dynamic rupture descriptions in Section 2.4.

2.1 Elasticity

Elastic rheological models are commonly used in seismology. In one dimension, the elastic model is equivalent to a linear spring, where stress and strain are related by a constant (Hooke's law). In multiple dimensions, a linear constitutive relation between stress and strain is most generally stated as

$$\sigma_{ij} = c_{ijkl}\epsilon_{kl}. \tag{2.5}$$

Due to symmetry considerations, the four-dimensional tensor c has only 21 independent coefficients [3, 145]. In the isotropic case, c simplifies to

$$c_{ijkl} = \lambda\delta_{ij}\delta_{kl} + \mu(\delta_{ik}\delta_{jl} + \delta_{il}\delta_{jk}), \tag{2.6}$$

where λ and μ are called Lamé parameters and $\delta_{ij} = 1$ whenever $i = j$ and zero otherwise. The second parameter μ is called the shear modulus and has the unit pascal (the strain tensor is dimensionless). The shear modulus directly relates the shear stress to an applied shear strain: One may show that $\sigma_{ij} = 2\mu e_{ij}$, where the deviatoric strain is defined as $e_{ij} = \epsilon_{ij} - \delta_{ij}\epsilon_{kk}/3$. (Note that for a pure shear strain we have $\epsilon_{ij} = e_{ij}$.) The parameter λ has the unit pascal, too, but the physical interpretation of λ is less clear. However, one may relate λ to the incompressibility modulus K with $K = \lambda + 2\mu/3$. The latter has the following interpretation [145]: Assume a unit cube is compressed from all sides with pressure p, then the stress tensor is given by $\sigma_{ij} = -p\delta_{ij}$ and the trace of the stress tensor is

$$\sigma_{kk} = -3p = 3\lambda\theta + 2\mu\theta, \tag{2.7}$$

where $\theta = \epsilon_{kk}$ is the dilatation (volume change) [145]. Thus, $-p = K\theta$ and as such K relates pressure to volume change.

A first order system of partial differential equations, which is usually called the elastic wave equation in velocity-stress form, is obtained by plugging (2.1) into the time derivative of (2.5). Together with (2.4), (2.6), and the definition $u_i := \frac{\partial}{\partial t} U_i$ one obtains

$$\begin{aligned} \frac{\partial}{\partial t}\sigma_{ij} - \lambda\delta_{ij}\frac{\partial}{\partial x_k}u_k - \mu\left(\frac{\partial}{\partial x_j}u_i + \frac{\partial}{\partial x_i}u_j\right) = 0, \\ \rho\frac{\partial}{\partial t}u_i - \frac{\partial}{\partial x_j}\sigma_{ji} = f_i. \end{aligned} \tag{2.8}$$

As the stress tensor is symmetric, the system of PDEs has 9 unknowns which are comprised of 6 stress components and 3 velocity components.

2.2 Viscoelasticity

Observations show that Earth is not elastic but seismic waves are attenuated over time [102]. It is assumed that attenuation is caused by a superposition of several physical mechanisms in natural rock [74, 102]. These physical mechanisms are modelled using the theory of linear viscoelasticity, such that the relation between stress and strain is frequency-dependent [47, 102]. In time-domain the relation between stress and strain becomes a convolution, which is intractable in numerical methods. Therefore, the standard method is to sample the relaxation spectrum with discrete relaxation mechanisms, where each discrete mechanism corresponds to a standard linear solid [23, 47, 51, 81, 102, 113].

In the remainder of this section, we briefly present how attenuation is incorporated into a system of PDEs.

2.2.1 Constitutive relation

For viscoelastic materials, one uses a constitutive relation in which the stress tensor depends upon the complete history of the strain tensor [25] (the dependence on location $\boldsymbol{x}$ is dropped in the following for the sake of readability):

$$\sigma_{ij}(t) = \int_{-\infty}^{t} G_{ijkl}(t-\tau)\frac{\partial \epsilon_{kl}(\tau)}{\partial \tau}\,\mathrm{d}\tau \tag{2.9}$$

where $\lim_{t\to-\infty}\epsilon_{ij}(t) = 0$. We introduce a few useful conventions: An asterisk indicates convolution, $f^c(t) = f(t)H(t)$ for a function f and the unit

step function H, and the dot above a symbol denotes the time derivative. With these conventions we abbreviate (2.9) as

$$\sigma_{ij}(t) = G^c_{ijkl} * \dot{\epsilon}_{kl} \tag{2.10}$$

Note that G^c_{ijkl} is causal, that is, stress depends only on current and past strain.

Due to symmetry considerations and isotropy, the fourth order tensor G may be expressed by two functions G_1 and G_2 in the following way [25]

$$G_{ijkl}(t) = \frac{1}{3}\left(G_2(t) - G_1(t)\right)\delta_{ij}\delta_{kl} + \frac{1}{2}G_1(t)\left(\delta_{ik}\delta_{jl} + \delta_{il}\delta_{jk}\right). \tag{2.11}$$

In analogy to the elastic case, we identify $\lambda = (G_2 - G_1)/3$, $\mu = G_1/2$, and $K = G_2/3$ (setting G_1 and G_2 to a constant value yields again an elastic constitutive relation). As in the elastic case, we obtain

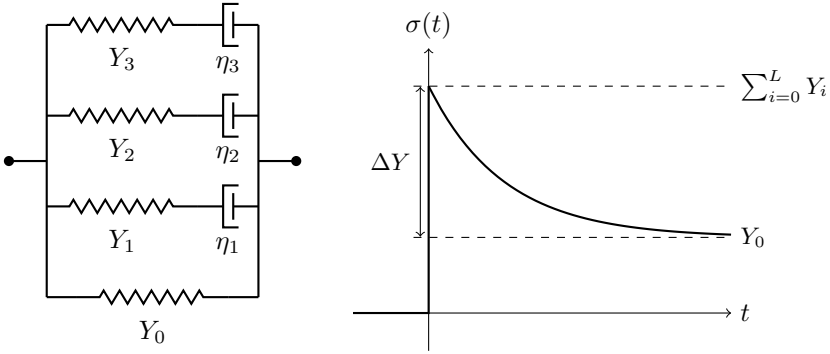
$$\begin{aligned} \sigma_{ij} &= \int_{-\infty}^{t} G_1(t-\tau)\frac{\partial e_{ij}(\tau)}{\partial \tau}\,\mathrm{d}\tau = G^c_1 * \dot{e}_{ij}, \\ \sigma_{kk} &= \int_{-\infty}^{t} G_2(t-\tau)\frac{\partial \theta(\tau)}{\partial \tau}\,\mathrm{d}\tau = G^c_2 * \dot{\theta}. \end{aligned} \tag{2.12}$$

The functions G_1 and G_2 are called relaxation functions, which becomes clear when inserting a unit jump H in the deviatoric strain tensor: With $e_{12} = H$ one obtains $\sigma_{12} = G^c_1 * \delta = G^c_1$ and with $\theta = H$ one obtains $\sigma_{kk} = G^c_2$. An example for a typical relaxation function is shown in Figure 2b.

A common ansatz in time-domain methods is to use the rheological model of a generalised Maxwell body (GMB) [47] or a generalised Zener body (GZB) [23, 102], which are equivalent [113]. For a GMB, the relaxation functions are given by

$$v = 1, 2 : G^c_v(t) = \left(Y_{0v} + \sum_{i=1}^{L} Y_{iv} e^{-\omega_{iv} t}\right) H(t). \tag{2.13}$$

A GMB consists of L parallel Maxwell bodies. Each Maxwell body consists of a spring with stiffness Y_{iv} in series with a damper with viscosity η_{iv}, leading to a relaxation time of $\omega_{iv}^{-1} = \eta_{iv}/Y_{iv}$. An additional parallel spring with stiffness Y_{0v} complements the L Maxwell bodies. Figure 2a shows a schematic of a GMB with $L = 3$.



(a) Generalised Maxwell body with 3 mechanisms.

(b) Response to unit step strain.

Figure 2: The sketches illustrate a generalised Maxwell body and stress response to a unit step strain. Figures adapted from Fig. 1 and Fig. 5 in [47].

2.2.2 Quality factor

The attenuation of seismic waves is commonly quantified by a frequency dependent quality factor Q. Physically, the inverse quality factor Q^{-1} can be thought of as energy loss per cycle. However, there is quite some confusion in the literature about the precise definition of Q [3, 13, 85, 117]. Nevertheless, authors end up with the relation [3, 13, 23, 47, 102, 117, 145]

$$Q = \frac{\mathrm{Re}(M(\omega))}{\mathrm{Im}(M(\omega))}, \tag{2.14}$$

where M is the complex modulus that is obtained for the constitutive relation in the frequency domain. Henceforth, we use (2.14) as definition of Q.

In order to obtain a relation between the quality factors and the relaxation functions defined in Section 2.2.1, we first transform (2.12) to the frequency domain. With a tilde denoting the Fourier transformation, that is $f^{\sim}(\omega) = \int_{\mathbb{R}} f(t)e^{-i\omega t}\,\mathrm{d}t$, we obtain

$$\begin{aligned}(\sigma_{ij})^{\sim} &= (\dot{G}_1^c)^{\sim}(e_{ij})^{\sim},\\ (\sigma_{kk})^{\sim} &= (\dot{G}_2^c)^{\sim}\theta^{\sim}.\end{aligned} \tag{2.15}$$

Using Equation (2.14) we get the quality factors

$$v = 1, 2 : Q_v = \frac{\mathrm{Re}[(\dot{G}_v^c)^\sim]}{\mathrm{Im}[(\dot{G}_v^c)^\sim]}. \tag{2.16}$$

We have obtained two quality factors as we have to distinguish between shear deformation and dilatation. However, in seismological applications it is more common to work with quality factors for P-waves and S-waves, which are given by [23]

$$Q_P = \frac{\mathrm{Re}[(\dot{G}_2^c)^\sim + 2(\dot{G}_1^c)^\sim]}{\mathrm{Im}[(\dot{G}_2^c)^\sim + 2(\dot{G}_1^c)^\sim]} \quad \text{and} \quad Q_S = \frac{\mathrm{Re}[(\dot{G}_1^c)^\sim]}{\mathrm{Im}[(\dot{G}_1^c)^\sim]}. \tag{2.17}$$

In the following we derive an explicit representation of the quality factors for the relaxation function of a GMB from (2.13). The first step is to derive the Fourier transformation of the relaxation functions. By using $(\dot{H})^\sim = (\delta)^\sim = 1$, $(\dot{f})^\sim = i\omega f^\sim$, and $(e^{-\alpha t}H)^\sim = 1/(\alpha + i\omega)$ we obtain

$$(\dot{G}_v^c)^\sim = (Y_{0v}\dot{H})^\sim + \sum_{i=1}^{L} i\omega (Y_{iv} e^{-\omega_{iv} t} H)^\sim = Y_{0v} + \sum_{i=1}^{L} \frac{i\omega Y_{iv}}{\omega_{iv} + i\omega}. \tag{2.18}$$

Note that we may have derived the last equality by summing up the impedances in Figure 2a, which shows again the connection between the relaxation function and the GMB model. The quality factors Q_1 and Q_2 are then given by

$$Q_\nu = \left(Y_{0v} + \sum_{i=1}^{L} \frac{Y_{iv}\omega^2}{\omega_{iv}^2 + \omega^2} \right) \left(\sum_{l=1}^{L} \frac{Y_{iv}\omega_{iv}\omega}{\omega_{iv}^2 + \omega^2} \right)^{-1} \tag{2.19}$$

and we may come up with similar expressions for Q_P and Q_S.

The explicit expression for Q in (2.19) shows that the GMB leads to Q being a rational function of ω, which is parameterised by stiffnesses Y_{iv}, with $i = 0, \ldots, L$, and relaxation frequencies ω_{iv}, with $i = 1, \ldots, L$. So if a Q-law is not given by a rational function, we cannot represent Q exactly with a GMB, but only approximately. By introducing a fitness measure, we obtain a non-linear optimisation problem in the $2L + 1$ free parameters. A simple but effective procedure to fit the free parameters is to choose logarithmically spaced relaxation frequencies and to determine the stiffnesses with a least squares problem [47]. An alternative approach based on Monte Carlo optimisation exists [51].

2.2.3 Memory variables

We need to integrate a convolution into a time-domain method. The standard trick is to model the relaxation spectrum with a GMB consisting of L Maxwell bodies. The relaxation function of a GMB, cf. (2.13), has a tractable form and one may reformulate the convolution integral as a system of PDEs. The general idea is to apply Leibniz's rule on the following special case:

$$\partial/\partial t \int_{-\infty}^{t} e^{-\omega(t-\tau)} \dot{\epsilon}_{ij}(\tau)\,\mathrm{d}\tau = \dot{\epsilon}_{ij} - \omega \int_{-\infty}^{t} e^{-\omega(t-\tau)} \dot{\epsilon}_{ij}(\tau)\,\mathrm{d}\tau. \tag{2.20}$$

By replacing the integral in above equation with a "memory variable" one may track the evolution of the integral with a differential equation.

To work out the details, we first introduce some shorthand notation. Motivated by the relation of λ and μ to G_1 and G_2 we define

$$Y^{\mu}_{\ i} = \frac{1}{2} Y_{i1} \quad \text{and} \quad Y^{\lambda}_{\ i} = \frac{1}{3}(Y_{l2} - Y_{l1}). \tag{2.21}$$

Moreover, we define the unrelaxed moduli (recall Figure 2b) as

$$\mu^U = \frac{1}{2} \sum_{l=0}^{L} Y_{l1} \quad \text{and} \quad \lambda^U = \frac{1}{3} \sum_{l=0}^{L} (Y_{l2} - Y_{l1}) \tag{2.22}$$

The relaxation frequencies are defined to be equal, i.e. $\omega_i := \omega_{i1} = \omega_{i2}$, which halves the number of memory variables.

The relaxation functions (2.13) are inserted into (2.11) and then into the constitutive relation (2.10):

$$\begin{aligned}
\sigma_{ij} &= \frac{1}{3}\left(G^c_2 - G^c_1\right) \delta_{ij} * \dot{\epsilon}_{kk} + G^c_1 * \dot{\epsilon}_{ij} \\
&= \left(Y^{\lambda}_{\ 0} + \sum_{l=1}^{L} Y^{\lambda}_{\ l}\, e^{-\omega_l t} \right) H \delta_{ij} * \dot{\epsilon}_{kk} + \left(2Y^{\mu}_{\ 0} + \sum_{l=1}^{L} 2Y^{\mu}_{\ l}\, e^{-\omega_l t} \right) H * \dot{\epsilon}_{ij} \\
&= Y^{\lambda}_{\ 0}\, \delta_{ij} \epsilon_{kk} + 2Y^{\mu}_{\ 0}\, \epsilon_{ij} + \sum_{l=1}^{L} Y^{\lambda}_{\ l}\, \delta_{ij} \int_{-\infty}^{t} e^{-\omega_l (t-\tau)} \dot{\epsilon}_{kk}(\tau)\,\mathrm{d}\tau \\
&\quad + \sum_{l=1}^{L} 2Y^{\mu}_{\ l} \int_{-\infty}^{t} e^{-\omega_l (t-\tau)} \dot{\epsilon}_{ij}(\tau)\,\mathrm{d}\tau,
\end{aligned} \tag{2.23}$$

Similar to the elastic case we are looking for a velocity-stress formulation, hence we take the time derivative of the stress tensor:

$$\dot{\sigma}_{ij} = \lambda^U \delta_{ij} \dot{\epsilon}_{kk} + 2\mu^U \dot{\epsilon}_{ij} - \sum_{l=1}^{L} Y^{\lambda}_{l}\, \delta_{ij} \omega_l \int_{-\infty}^{t} e^{-\omega_l (t-\tau)} \dot{\epsilon}_{kk}(\tau)\, \mathrm{d}\tau - \sum_{l=1}^{L} 2Y^{\mu}_{l}\, \omega_l \int_{-\infty}^{t} e^{-\omega_l (t-\tau)} \dot{\epsilon}_{ij}(\tau)\, \mathrm{d}\tau, \quad (2.24)$$

where we used Leibniz's rule. Our memory variables are defined as

$$\zeta_{ijl}(t) = \omega_l \int_{-\infty}^{t} e^{-\omega_l (t-\tau)} \dot{\epsilon}_{ij}(\tau)\, \mathrm{d}\tau, \quad (2.25)$$

which satisfy the differential equation

$$\dot{\zeta}_{ijl} = \omega_l \dot{\epsilon}_{ij} - \omega_l \zeta_{ijl}. \quad (2.26)$$

Note that no sum for l is implied in the last term. We obtain the final relation between stress-rate and strain-rate,

$$\dot{\sigma}_{ij} = \lambda^U \delta_{ij} \dot{\epsilon}_{kk} + 2\mu^U \dot{\epsilon}_{ij} - \sum_{l=1}^{L} Y^{\lambda}_{l}\, \delta_{ij} \zeta_{kkl} - \sum_{l=1}^{L} 2Y^{\mu}_{l}\, \zeta_{ijl}, \quad (2.27)$$

by inserting the memory variables into (2.24).

The full set of equations in velocity-stress form consists of the constitutive relation, Newton's second law, and the memory variable equations. Similar to the elastic case, the time derivative of strain is replaced by particle velocities, such that the following system of partial differential equations is obtained:

$$\frac{\partial}{\partial t}\sigma_{ij} - \lambda^U \delta_{ij} \frac{\partial}{\partial x_k} u_k - \mu^U \left(\frac{\partial}{\partial x_j} u_i + \frac{\partial}{\partial x_i} u_j \right) = -\sum_{l=1}^{L} (Y^{\lambda}_{l}\, \delta_{ij} \zeta_{kkl} + 2Y^{\mu}_{l}\, \zeta_{ijl}), \quad (2.28)$$

$$\rho \frac{\partial}{\partial t} u_i - \frac{\partial}{\partial x_j} \sigma_{ji} = f_i, \quad (2.29)$$

$$\frac{\partial}{\partial t}\zeta_{ijl} - \frac{\omega_l}{2} \left(\frac{\partial}{\partial x_j} u_i + \frac{\partial}{\partial x_i} u_j \right) = -\omega_l \zeta_{ijl}. \quad (2.30)$$

We need to store 6 stress components and 3 velocity components. In addition to the elastic case, 6 memory variables per mechanism are required, such that we require $9 + 6L$ quantities in total. A typical choice for the number of mechanisms is $L = 3$.

2.3 Acoustics, Anisotropy, and Plasticity

The focus of this thesis lies on the elastic and viscoelastic rheological models. Nevertheless, we briefly mention other rheological models, which also fit in the framework of the ADER-DG method.

A special case of the isotropic elastic wave equation are the equations of linearised acoustics. These may be obtained by setting the shear modulus μ and the shear stresses to zero [80].

Another straightforward extension of the isotropic elastic model is the anisotropic elastic model. One needs to specify the 21 independent coefficients of the tensor c_{ijkl} which links stress to strain. The derivation of the system of PDEs is then equivalent to Section 2.1.

In the vicinity of geological faults, high stresses may occur during an earthquake, such that a linear elastic model becomes inappropriate. One needs to account for plastic deformation, which may be modelled with Drucker-Prager plasticity [164]: The total strain is divided into elastic strain ϵ^e and viscoplastic strain ϵ^{vp}, such that $\epsilon = \epsilon^e + \epsilon^{vp}$. The stress-strain relation is given by

$$\sigma_{ij} = c_{ijkl}(\epsilon_{kl} - \epsilon^{vp}_{kl}). \tag{2.31}$$

The viscoplastic strain increases with stress whenever stress exceeds a yield function F, that is, whenever $F(\sigma) \geq 0$. In the isotropic case, the viscoplastic strain rate is given by [164]

$$\dot{\epsilon}^{vp}_{ij} = \frac{1}{2\mu T_v}(\sigma_{ij} - P_{ij}(\sigma)), \tag{2.32}$$

for a positive relaxation time T_v. The right-hand side of (2.32) is only non-zero if $F(\sigma) \geq 0$. In essence, plasticity as described here can be seen as a non-linear source term, as neither additional PDEs are introduced nor the viscoplastic strain rate depends on derivatives of the stress tensor.

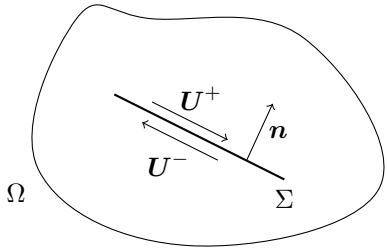


Figure 3: Schematic depiction of a fault, modelled as an internal surface Σ in volume Ω. An earthquake is caused by slip on a fault, which is modelled as displacement discontinuity $\boldsymbol{U}^+ - \boldsymbol{U}^-$.

2.4 Earthquake sources

A geological fault is a zone were rock is fractured or weakened. The fault is often idealised as a plane, or mathematically an internal surface Σ embedded in a volume Ω. (See Figure 3 for a sketch in two dimensions.) In the locked state, stress builds up on the fault – e.g. due to tectonics – until the stress exceeds the fault's strength. Then, the two sides of the idealised fault plane loose contact and start slipping, which releases stored energy in form of seismic waves. The slipping faults lead to a discontinuity in the displacements, where the difference in displacement is $[\![\boldsymbol{U}]\!] := \boldsymbol{U}^+ - \boldsymbol{U}^-$. The displacements $\boldsymbol{U}^\pm$ are defined as the limiting values at the interface, that is, $\boldsymbol{U}^\pm(\boldsymbol{x}) = \lim_{\epsilon\to 0} \boldsymbol{U}(\boldsymbol{x} \pm \epsilon \boldsymbol{n})$. The magnitude of $[\![\boldsymbol{U}]\!]$ is called slip.

Two distinct approaches to model rupture on a fault are discussed in the following.

2.4.1 Kinematic rupture

The equations of motion are invalid on the surface Σ, due to the displacement discontinuity. As a remedy, one may derive equivalent body forces for a given fault surface and a given slip distribution, which give an equivalent displacement in the volume. It can be shown that equivalent body forces are given by the following equation [3, Equation 3.5]:

$$f_p(\boldsymbol{x}, t) = -\int_\Sigma [\![U_i(\boldsymbol{\xi}, t)]\!] c_{ijpq} n_j \frac{\partial}{\partial x_q} \delta(\boldsymbol{x} - \boldsymbol{\xi}) \, \mathrm{d}\Sigma, \tag{2.33}$$

where $\boldsymbol{n}$ is the fault surface's normal and $\boldsymbol{\xi}$ is the position on Σ. In the case that the wavelengths of observed waves become much larger than the dimensions of Σ, it is sufficient to condense the slip into a single point [3]. Hence, we consider slip given by

$$[\![U_i]\!](\boldsymbol{\xi}, t) = AS_i(t)\delta(\boldsymbol{\xi} - \boldsymbol{x}_c). \tag{2.34}$$

The time history of slip are functions $S_i(t)$ which are scaled with the area of the fault A, and the source acts on a point $\boldsymbol{x}_c$, say the fault's centre. Inserting the slip into (2.33) we obtain

$$f_p(\boldsymbol{x}, t) = -\int_\Sigma AS_i(t)\delta(\boldsymbol{\xi} - \boldsymbol{x}_c)c_{ijpq}n_j \frac{\partial}{\partial x_q}\delta(\boldsymbol{x} - \boldsymbol{\xi})\,\mathrm{d}\Sigma. \tag{2.35}$$

The body force becomes a convolution of two delta distributions, and thus[2]

$$f_p(\boldsymbol{x}, t) = -AS_i(t)c_{ijpq}n_j \frac{\partial}{\partial x_q}\delta(\boldsymbol{x} - \boldsymbol{x}_c). \tag{2.36}$$

In seismology, one commonly specifies kinematic rupture sources in terms of a moment tensor, which sums up the body force terms but excludes the delta distribution [3]:

$$M_{pq}(t) = AS_i(t)c_{ijpq}n_j. \tag{2.37}$$

Inserting the constitutive relation of an isotropic medium, the moment tensor becomes

$$M_{pq} = \lambda AS_i n_i \delta_{pq} + \mu A(S_p n_q + S_q n_p). \tag{2.38}$$

Plugging the body forces into the second equation of (2.8) we get

$$\rho\frac{\partial}{\partial t}u_i - \frac{\partial}{\partial x_j}\sigma_{ji} = -M_{iq}(t)\frac{\partial}{\partial x_q}\delta(\boldsymbol{x} - \boldsymbol{x}_c). \tag{2.39}$$

[2] Using the definition of convolution for distributions, we have $\left\langle \frac{\partial}{\partial x_q}\delta(\boldsymbol{x}) * \delta(\boldsymbol{x} - \boldsymbol{x}_c), \phi(\boldsymbol{x}) \right\rangle = \left\langle \frac{\partial}{\partial x_q}\delta(\boldsymbol{x}), \langle \delta(\boldsymbol{\xi} - \boldsymbol{x}_c), \phi(\boldsymbol{\xi} + \boldsymbol{x}) \rangle \right\rangle = -\frac{\partial}{\partial x_q}\phi(\boldsymbol{x}_c)$. The latter is identified as the distribution $\frac{\partial}{\partial x_q}\delta(\boldsymbol{x} - \boldsymbol{x}_c)$.

However in SeisSol, the moment tensor is not applied to Newton's second law, but to the stress strain relation [83]. To do so, one defines the modified stress tensor

$$\tilde{\sigma}_{ij} = \sigma_{ij} - M_{ji}\delta(\boldsymbol{x} - \boldsymbol{x}_c) \tag{2.40}$$

which leads to the PDEs

$$\begin{aligned} \frac{\partial}{\partial t}\tilde{\sigma}_{ij} - \lambda\delta_{ij}\frac{\partial}{\partial x_k}u_k - \mu\left(\frac{\partial}{\partial x_j}u_i + \frac{\partial}{\partial x_i}u_j\right) &= \frac{\partial}{\partial t}M_{ji}\delta(\boldsymbol{x} - \boldsymbol{x}_c), \\ \rho\frac{\partial}{\partial t}u_i - \frac{\partial}{\partial x_j}\tilde{\sigma}_{ji} &= 0. \end{aligned} \tag{2.41}$$

The technical discussion so far shows how an earthquake modelled as displacement discontinuity $[\![\boldsymbol{U}]\!]$ can be included in the wave equations. In reality the exact spatial distribution and temporal evolution of $[\![\boldsymbol{U}]\!]$ is unknown, which leads to the question of how the displacement on the fault of an earthquake event can be obtained, and what can be learnt from such models.

The displacements are often obtained from source inversion procedures. For many events, such inversion results can be found in the SRCMOD online database [107]. But, the inversion process is not unique. In particular, the authors of a recent study, which is concerned with the validation of source inversion, conclude that "Significantly different kinematic source models may explain the data equally well." [108]

2.4.2 Dynamic rupture

Dynamic rupture is an alternative modelling approach for earthquake sources. Instead of looking for equivalent body forces, boundary conditions for the fault surface Σ are introduced. These boundary conditions model frictional contact, such that the fault is in a locked state when the shear traction is smaller than the fault strength, and active slip when the shear traction is equal to the fault strength.

The conditions on the fault's surface are given by Day et al. [34], and are repeated here for reference. The traction vector is given by $\boldsymbol{t} = \sigma\boldsymbol{n}$ ($\boldsymbol{n}$ is the unit normal of Σ), and the shear traction is the tangential component of traction, i.e. $\boldsymbol{\tau} := \boldsymbol{t} - (\boldsymbol{n}\cdot\boldsymbol{t})\boldsymbol{n}$. The shear traction is bound by the fault strength τ_S,

$$\|\boldsymbol{\tau}\| \leq \tau_S. \tag{2.42}$$

The slip rate is given by $[\![\boldsymbol{u}]\!]$. The shear traction is related to the tangential part of the slip rate, which is $\boldsymbol{s} := [\![\boldsymbol{u}]\!] - (\boldsymbol{n} \cdot [\![\boldsymbol{u}]\!])\boldsymbol{n}$, in the following way:

$$\tau_S \boldsymbol{s} - \boldsymbol{\tau} \left\| \boldsymbol{s} \right\| = 0. \tag{2.43}$$

Equations (2.42) and (2.43) imply that the shear traction may not exceed the fault strength, that the slip rate and the shear traction are anti-parallel, and that the slip rate must be zero whenever the shear traction is not equal to the fault strength [34].

The fault strength evolves according to

$$\begin{aligned} \tau_S &= \max\left(0, -\sigma_n f\left(\left\| \boldsymbol{s} \right\|, \psi\right)\right), \\ \frac{\mathrm{d}\psi}{\mathrm{d}t} &= g\left(\left\| \boldsymbol{s} \right\|, \psi\right), \end{aligned} \tag{2.44}$$

where $\sigma_n := \boldsymbol{n} \cdot \boldsymbol{t}$ is the normal stress (negative in compression), the function f is the friction law, and ψ is the state variable. For example, in the linear slip-weakening model [34, 71], the coefficient of friction decreases with increasing slip path length, i.e.

$$\begin{aligned} f\left(\left\| \boldsymbol{s} \right\|, \psi\right) &= \mu_s - \frac{\mu_s - \mu_d}{d_C} \min(\psi, d_c), \\ g\left(\left\| \boldsymbol{s} \right\|, \psi\right) &= \left\| \boldsymbol{s} \right\|, \end{aligned} \tag{2.45}$$

where $\psi(0) = 0$. The static coefficient of friction μ_s, the dynamic coefficient of friction μ_d, and the critical distance d_c are the parameters of the friction law. Note that $\psi = \int_0^t \left\| \boldsymbol{s}(t') \right\| \mathrm{d}t'$, that is, the state variable contains the slip path length.

Dynamic rupture is a potential method to gain insight into earthquake source mechanics, as the model parameters are physically motivated. A difficulty with dynamic rupture models is that no analytical solutions for the general three-dimensional case exists [34], such that it is difficult to verify numerical methods implementing dynamic rupture. Indirect verification methods exist, such as comparison to analytical solutions for simplified problems [89], the method of manufactured solutions [42], or comparison of numerical solutions among various codes [61, 62]. Due to the difficult verification process, additional numerical analysis is imperative, hence a stability analysis of the dynamic rupture method in SeisSol [123] is contributed in Chapter 5.

CHAPTER

3

ADER-DG in a nutshell

The common denominator of all systems of PDEs in Chapter 2 is that they are linear hyperbolic (with the exception of plasticity). Thus, throughout this chapter we assume that a general system of linear hyperbolic PDEs is given and has the following form:

$$\frac{\partial q_p}{\partial t} + A_{pq}{}^{1} \frac{\partial q_q}{\partial x_1} + A_{pq}{}^{2} \frac{\partial q_q}{\partial x_2} + A_{pq}{}^{3} \frac{\partial q_q}{\partial x_3} = E_{pq} q_q, \tag{3.1}$$

where q is the vector of quantities, $A_{pq}{}^{1}$, $A_{pq}{}^{2}$, and $A_{pq}{}^{3}$ are coefficient matrices, and E_{pq} is the matrix of a linear source term. The coefficient matrices are assumed to be space-dependent, hence (3.1) is given in non-conservative form [100, Chapter 9]. Note that we use the Einstein convention, that is, indices appearing twice are implicitly summed over, unless the opposite is noted.

The original research on the Arbitrary high-order DERivatives Discontinuous Galerkin (ADER-DG) method in computational seismology is presented in [39, 41, 81]. In the following, ADER-DG is briefly summarised in order to make this document self-contained. For a general introduction to DG, the reader is referred to [68]. Some aspects, such as the parameterisation of surface integrals, are covered in more detail. The author's hope is that this chapter is a useful supplement to the original papers.

Figure 4: Index placement conventions for the element index, element-local indices, and additional indices.

Notation. We counteract the inevitable index nightmare by introducing the following conventions for tensors used throughout this thesis. In the north west of a tensor symbol, cf. Figure 4, we write the *element index*. The element index might be dropped when the element is clear from the context. The *element-local indices* are placed in the south east of a tensor symbol, and *additional indices* are placed in the north east of a tensor symbol. For example the 4D tensor ${}^{m}Q_{lp}{}^{n}$ has the element-local indices lp and we may think of it as an element-local matrix. Sometimes the element-local indices are dropped, e.g. we write ${}^{m}\boldsymbol{\xi}^{a}$ for the tensor ${}^{m}\xi_{i}{}^{a}$. The bold-face symbol then represents an (element-local) vector.

3.1 Discretisation in space

The domain of interest Ω is approximated by a conforming tetrahedral mesh with tetrahedra $\mathcal{T}_m$. Multiplying (3.1) by a test function ϕ_k and integrating over the tetrahedron $\mathcal{T}_m$ we obtain

$$\int_{\mathcal{T}_m} \phi_k \frac{\partial q_p}{\partial t} \,\mathrm{d}\boldsymbol{x} + \int_{\mathcal{T}_m} \phi_k A_{pq}{}^{d} \frac{\partial q_q}{\partial x_d} \,\mathrm{d}\boldsymbol{x} = \int_{\mathcal{T}_m} \phi_k E_{pq} q_q \,\mathrm{d}\boldsymbol{x}. \tag{3.2}$$

The space-dependent tensor $A_{pq}{}^{d}$ is approximated to be constant on tetrahedron $\mathcal{T}_m$. By integrating the second integral by parts we obtain the weak form

$$\int_{\mathcal{T}_m} \phi_k \frac{\partial q_p}{\partial t} \,\mathrm{d}\boldsymbol{x} + \int_{\partial\mathcal{T}_m} \phi_k \left(n_d A_{pq}{}^{d} q_q\right)^{*} \mathrm{d}S - \int_{\mathcal{T}_m} \frac{\phi_k}{\partial x_d} A_{pq}{}^{d} q_q \,\mathrm{d}\boldsymbol{x} = \int_{\mathcal{T}_m} \phi_k E_{pq} q_q \,\mathrm{d}\boldsymbol{x}, \tag{3.3}$$

where $\boldsymbol{n} = (n_1, n_2, n_3)$ is the outward pointing unit surface normal vector. The boundary values appear always twice as each face is shared by two tetrahedra (except at the domain boundary). A numerical flux is introduced – marked with a star in (3.3) – to weakly couple the values

Table 1: Face local vertex index to tetrahedron local vertex index mapping [39].

j	$\nu_0(j)$	$\nu_1(j)$	$\nu_2(j)$	$\nu_3(j)$
0	0	0	0	1
1	2	1	3	2
2	1	3	2	3

at the interface. The role of the numerical flux is to ensure stability of the numerical scheme [68]. In Chapter 4, the numerical flux is covered in detail. For now we assume that the numerical flux may be written in the following form for the f-th face of a tetrahedron:

$$\left(n_d A_{pq}{}^d q_q\right)^* = \mathcal{A}^+{}_{pq}{}^f(\boldsymbol{n}) \lim_{\epsilon \to 0} q_q(\boldsymbol{x} - \epsilon \boldsymbol{n}, t) + \mathcal{A}^-{}_{pq}{}^f(\boldsymbol{n}) \lim_{\epsilon \to 0} q_q(\boldsymbol{x} + \epsilon \boldsymbol{n}, t) \tag{3.4}$$

The numerical flux handles discontinuities by incorporating information from both sides of an interface.

3.1.1 Unstructured mesh

We formally define the tetrahedral mesh to be used in the spatial discretisation.

The mesh consists of a list of V vertices $\boldsymbol{x}^0, \ldots, \boldsymbol{x}^{V-1}$ (in $\mathbb{R}^3$) and a connectivity function ${}^m c : \{0, \ldots, 3\} \to \{0, \ldots, V-1\}$ for each tetrahedron m. The connectivity function takes the tetrahedron's local vertex index as argument and maps it to the global vertex index. Thus the four vertices of tetrahedron m are given by $\boldsymbol{x}^{{}^m c(j)}$, $j = 0, \ldots, 3$.

The DG scheme requires surface integrals, which we need to parameterise for evaluation, and hence we need to carefully define the mapping from faces to elements: For the f-th face of a tetrahedron the function $\nu_f : \{0, ..., 2\} \to \{0, \ldots, 3\}$ maps the face's local vertex index to the tetrahedron's local vertex index. We choose the convention from Dumbser et al. [39] for the functions ν_f, which is summarised in Table 1.

When an element has a neighbour element they must share three vertices. Let the set of vertices of element m be ${}^m\mathcal{C} := {}^m c(\{0, \ldots, 3\})$. Then, element m shares a face with element n if $|{}^m\mathcal{C} \cap {}^n\mathcal{C}| = 3$. Due to the latter property, we can establish a relation between the connectivity functions of elements m and n. The shared face corresponds to a local face index

in the respective elements, say f is the local index of the shared face in element m and g is the local index of the shared face in element n. Then the image of the functions ${}^{m}c \circ \nu_f$ and ${}^{n}c \circ \nu_g$ is identical. As the domain of the functions is identical, too, we conclude that

$$ {}^{m}c(\nu_f(j)) = {}^{n}c(\nu_g(\pi_h(j))), \quad \forall j \in \{0, \ldots, 2\}, \tag{3.5} $$

where π_h is a permutation function.

The permutation function π_h may be further constrained. Obviously, there are at most 3! permutation functions on a domain of cardinality 3. Furthermore, we presume that the mesh generator orders the face's local vertex indices in strictly counter-clockwise order, which halves the number of possible permutation functions. All three possible cases are illustrated in Figure 5. We infer that the permutation function is given by

$$ \pi_h(j) = (3 + h - j) \bmod 3, \tag{3.6} $$

where h is chosen such that ${}^{m}c(\nu_f(0)) = {}^{n}c(\nu_g(h))$.

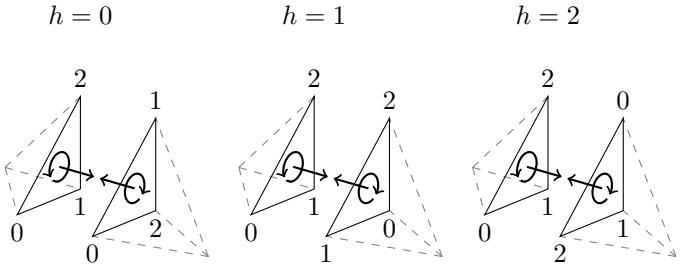


Figure 5: Neighbour configurations of tetrahedra. Due to the counter-clockwise ordering of local vertex indices, only three configurations exist.

3.1.2 Basis functions and volume integrals

In each tetrahedron, the quantities are approximated with polynomials of maximum degree N. As the space of polynomials with maximum degree N is finite dimensional, every quantity may be written as a finite basis expansion:

$$ q_p(\boldsymbol{x}, t) = {}^{m}Q_{lp}(t)\phi_l\left({}^{m}\boldsymbol{\Xi}(\boldsymbol{x})\right). \tag{3.7} $$

The basis functions $\phi_l(\boldsymbol{\xi})$ are defined on the reference tetrahedron $\mathcal{E}_3$ with vertices $\{\mathbf{0}, \boldsymbol{e}^1, \boldsymbol{e}^2, \boldsymbol{e}^3\}$, where ${e_j}^i = \delta_{ij}$. We choose the test functions identical to the basis functions, which is commonly known as Galerkin approach [68]. The function ${}^m\boldsymbol{\Xi}$ maps from tetrahedron m to the reference tetrahedron. Given the four vertices $\{\boldsymbol{x}^{{}^m c(0)}, \boldsymbol{x}^{{}^m c(1)}, \boldsymbol{x}^{{}^m c(2)}, \boldsymbol{x}^{{}^m c(3)}\}$ of tetrahedron m, the map is given by

$$ {}^m\boldsymbol{\Xi}(\boldsymbol{x}) = {}^m\Theta^{-1}\left(\boldsymbol{x} - \boldsymbol{x}^{{}^m c(0)}\right). \tag{3.8} $$

Henceforth, we drop the superscript m for brevity, but the reader should keep the dependency on element m in mind. The matrix Θ is defined as

$$ \Theta = \begin{pmatrix} \boldsymbol{x}^{c(1)} - \boldsymbol{x}^{c(0)} & \boldsymbol{x}^{c(2)} - \boldsymbol{x}^{c(0)} & \boldsymbol{x}^{c(3)} - \boldsymbol{x}^{c(0)} \end{pmatrix}. \tag{3.9} $$

The map $\boldsymbol{\Xi}$ is best understood by its inverse map $\boldsymbol{X} := \boldsymbol{\Xi}^{-1}$,

$$ \boldsymbol{X}(\boldsymbol{\xi}) := \Theta\boldsymbol{\xi} + \boldsymbol{x}^{c(0)}, \tag{3.10} $$

with the properties $\boldsymbol{X}(\boldsymbol{e}^k) = \boldsymbol{x}^{c(k)}$, $\boldsymbol{X}(\mathbf{0}) = \boldsymbol{x}^{c(0)}$, and $\boldsymbol{X}(\mathcal{E}_3) = \mathcal{T}_m$. The map $\boldsymbol{X}$ is used later on to rewrite integrals over $\mathcal{T}_m$ as integrals over the reference element $\mathcal{E}_3$. That is, using the substitution rule we have

$$ \int_{\boldsymbol{X}(\mathcal{E}_3)} F(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x} = \int_{\mathcal{E}_3} F(\boldsymbol{X}(\boldsymbol{\xi}))|\det(D\boldsymbol{X})(\boldsymbol{\xi})|\,\mathrm{d}\boldsymbol{\xi} = \int_{\mathcal{E}_3} F(\boldsymbol{X}(\boldsymbol{\xi}))|J|\,\mathrm{d}\boldsymbol{\xi}, \tag{3.11} $$

for a function F, where $|J| = |\det(D\boldsymbol{X})(\boldsymbol{\xi})| = |\det(\Theta)|$.

The gradient of the test functions $\phi_k(\boldsymbol{\Xi}(\boldsymbol{x}))$ in (3.3) has to be taken w.r.t. the physical coordinates. Using the chain rule and (3.8) we compute the gradient w.r.t. the reference coordinates as following:

$$ \frac{\partial\phi_k}{\partial x_d} = \frac{\partial\phi_k}{\partial\xi_e}\frac{\partial\xi_e}{\partial x_d} = \frac{\partial\phi_k}{\partial\xi_e}\Theta^{-1}_{ed} \tag{3.12} $$

Next, we insert the finite basis expansion for q_p in the weak form (3.3). In combination with (3.12) and the substitution rule we obtain

$$ \begin{aligned} &\int_{\mathcal{E}_3} \phi_k(\boldsymbol{\xi})\frac{\partial Q_{lp}(t)}{\partial t}\phi_l(\boldsymbol{\xi})|J|\,\mathrm{d}\boldsymbol{\xi} + \int_{\partial\mathcal{T}_m} \phi_k(\boldsymbol{\Xi}(\boldsymbol{x}))(n_d {A_{pq}}^d q_q)^*\,\mathrm{d}S \\ &-\int_{\mathcal{E}_3} \frac{\phi_k(\boldsymbol{\xi})}{\partial\xi_e}\Theta^{-1}_{ed}{A_{pq}}^d Q_{lq}(t)\phi_l(\boldsymbol{\xi})|J|\,\mathrm{d}\boldsymbol{\xi} = \int_{\mathcal{E}_3} \phi_k(\boldsymbol{\xi})E_{pq}Q_{lq}(t)\phi_l(\boldsymbol{\xi})|J|\,\mathrm{d}\boldsymbol{x}. \end{aligned} \tag{3.13} $$

The above equation can be further simplified. As only the basis functions depend on space, all other terms may be written in front of the integral:

$$|J|\frac{\partial Q_{lp}(t)}{\partial t}\int_{\mathcal{E}_3}\phi_k\phi_l\,\mathrm{d}\boldsymbol{\xi}+\int_{\partial\mathcal{T}_m}\phi_k(n_d A_{pq}{}^d q_q)^*\,\mathrm{d}S$$
$$-|J|Q_{lq}(t)\Theta^{-1}_{ed}A_{pq}{}^d\int_{\mathcal{E}_3}\frac{\phi_k}{\partial\xi_e}\phi_l\,\mathrm{d}\boldsymbol{\xi}=|J|E_{pq}Q_{lq}(t)\int_{\mathcal{E}_3}\phi_k\phi_l\,\mathrm{d}\boldsymbol{x}. \quad (3.14)$$

In the above equation, the volume integral became independent of the physical coordinates of tetrahedron $\mathcal{T}_m$, such that the integrals on the reference tetrahedron may be precomputed.

3.1.3 Surface integrals and numerical flux

We aim to parameterise the surface integrals over a function F:

$$\int_{\partial\mathcal{T}_m}F\,\mathrm{d}S=\sum_{f=0}^{3}\int_{\mathcal{E}_2}F({}^m\boldsymbol{\Upsilon}^f(\boldsymbol{\chi}))\left\|\frac{\partial{}^m\boldsymbol{\Upsilon}^f}{\partial\chi_1}\times\frac{\partial{}^m\boldsymbol{\Upsilon}^f}{\partial\chi_2}\right\|\mathrm{d}\boldsymbol{\chi}, \quad (3.15)$$

where $\mathcal{E}_2$ is the reference triangle with vertices $\{(0,0),(1,0),(0,1)\}$. The functions ${}^m\boldsymbol{\Upsilon}^f$ are affine linear maps, which depend on the three vertices of a face f. These are $\boldsymbol{x}^{{}^mc(\nu_f(0))}$, $\boldsymbol{x}^{{}^mc(\nu_f(1))}$, and $\boldsymbol{x}^{{}^mc(\nu_f(2))}$. To simplify the notation, we introduce ${}^mc_f:={}^mc\circ\nu_f$ and drop the superscript m in the following.

The construction of the face parameterisations is very similar to the mapping function $\boldsymbol{X}$. We define them as

$$\boldsymbol{\Upsilon}^f(\boldsymbol{\chi})=(\boldsymbol{x}^{c_f(1)}-\boldsymbol{x}^{c_f(0)})\chi_1+(\boldsymbol{x}^{c_f(2)}-\boldsymbol{x}^{c_f(0)})\chi_2+\boldsymbol{x}^{c_f(0)}. \quad (3.16)$$

Using the definition of ν (cf. Table 1), one can easily show that above equation may be rewritten in the following way:

$$\boldsymbol{\Upsilon}^0(\boldsymbol{\chi})=\Theta\begin{pmatrix}0&1\\1&0\\0&0\end{pmatrix}\boldsymbol{\chi}+\boldsymbol{x}^0\quad\boldsymbol{\Upsilon}^1(\boldsymbol{\chi})=\Theta\begin{pmatrix}1&0\\0&0\\0&1\end{pmatrix}\boldsymbol{\chi}+\boldsymbol{x}^0$$
$$\boldsymbol{\Upsilon}^2(\boldsymbol{\chi})=\Theta\begin{pmatrix}0&0\\0&1\\1&0\end{pmatrix}\boldsymbol{\chi}+\boldsymbol{x}^0\quad\boldsymbol{\Upsilon}^3(\boldsymbol{\chi})=\Theta\begin{pmatrix}-1&-1\\1&0\\0&1\end{pmatrix}\boldsymbol{\chi}+\Theta\begin{pmatrix}1\\0\\0\end{pmatrix}+\boldsymbol{x}^0 \quad (3.17)$$

Table 2: Change of coordinates in surface integrals [39].

$\boldsymbol{\chi}$	$\tilde{\boldsymbol{\chi}}^0(\boldsymbol{\chi})$	$\tilde{\boldsymbol{\chi}}^1(\boldsymbol{\chi})$	$\tilde{\boldsymbol{\chi}}^2(\boldsymbol{\chi})$
χ_1	χ_2	$1-\chi_1-\chi_2$	χ_1
χ_2	χ_1	χ_2	$1-\chi_1-\chi_2$

Plugging in the face parameterisation in the basis functions, i.e. $\phi_l\left(\boldsymbol{\Xi}(\boldsymbol{x})\right)$, requires the composition $\boldsymbol{\Xi}(\boldsymbol{\Upsilon}^f)$. Using (3.17), the composition becomes

$$\begin{aligned}
\boldsymbol{\Xi}(\boldsymbol{\Upsilon}^0(\boldsymbol{\chi})) &= \begin{pmatrix}\chi_2\\ \chi_1\\ 0\end{pmatrix} \quad \boldsymbol{\Xi}(\boldsymbol{\Upsilon}^1(\boldsymbol{\chi})) = \begin{pmatrix}\chi_1\\ 0\\ \chi_2\end{pmatrix}\\
\boldsymbol{\Xi}(\boldsymbol{\Upsilon}^2(\boldsymbol{\chi})) &= \begin{pmatrix}0\\ \chi_2\\ \chi_1\end{pmatrix} \quad \boldsymbol{\Xi}(\boldsymbol{\Upsilon}^3(\boldsymbol{\chi})) = \begin{pmatrix}1-\chi_1-\chi_2\\ \chi_1\\ \chi_2\end{pmatrix}
\end{aligned} \tag{3.18}$$

Note that the composition is independent of the element's geometry. We introduce the shorthand notation $\boldsymbol{\xi}^f(\boldsymbol{\chi}) := \boldsymbol{\Xi}(\boldsymbol{\Upsilon}^f(\boldsymbol{\chi}))$ for the remainder of this section.

The numerical flux also involves data of the neighbouring element n, such that we additionally need the composition ${}^n\boldsymbol{\Xi}({}^m\boldsymbol{\Upsilon}^f)$. The trick here is to insert (3.5) into (3.16), and to introduce a change of coordinates $\tilde{\chi}$, such that ${}^m\boldsymbol{\Upsilon}^f(\boldsymbol{\chi}) = {}^n\boldsymbol{\Upsilon}^g\left(\tilde{\boldsymbol{\chi}}^h(\boldsymbol{\chi})\right)$, and thus ${}^n\boldsymbol{\Xi}({}^m\boldsymbol{\Upsilon}^f(\boldsymbol{\chi})) = \boldsymbol{\xi}^g(\tilde{\boldsymbol{\chi}}^h(\boldsymbol{\chi}))$. Through tedious but simple manipulations, one can show that $\tilde{\boldsymbol{\chi}}^h$ exists and does not depend on the geometry of either element m or element n. The function $\tilde{\boldsymbol{\chi}}$ is summarised in Table 2.

We now have all ingredients to compute the surface integrals. We assume that the numerical flux is defined as in (3.4). Taking the limit from positive and negative normal direction, we see that the flux incorporates information from element m and its four neighbouring elements, which we denote with m_f. So, the numerical flux is

$$(n_d A_{pq}{}^d q_q)^* = \mathcal{A}^+_{pq}{}^f \phi_l({}^m\boldsymbol{\Xi}(\boldsymbol{x}))\,{}^mQ_{lq}(t) + \mathcal{A}^-_{pq}{}^f \phi_l({}^{m_f}\boldsymbol{\Xi}(\boldsymbol{x}))\,{}^{m_f}Q_{lq}(t) \tag{3.19}$$

We parameterise each side of element m with the maps ${}^m\boldsymbol{\Upsilon}^f$. As it is shown, the compositions ${}^m\boldsymbol{\Xi}({}^m\boldsymbol{\Upsilon}^f)$ and ${}^{m_f}\boldsymbol{\Xi}({}^m\boldsymbol{\Upsilon}^f)$ become independent of element m or m_f and only depend on the face relation parameters,

which we call g_f and h_f. Thus, the following expression for the surface integrals is obtained:

$$\begin{aligned}\int_{\partial\mathcal{T}_m} \phi_k({}^m\boldsymbol{\Xi}(\boldsymbol{x}))(n_d A_{pq}{}^d q_q)^* \,\mathrm{d}S &= \sum_{f=0}^{3} \int_{\partial(\mathcal{T}_m)_f} \phi_k({}^m\boldsymbol{\Xi}(\boldsymbol{x})) \\ &\times \left(\mathcal{A}^+_{pq}{}^f \phi_l({}^m\boldsymbol{\Xi}(\boldsymbol{x}))^m Q_{lq}(t) + \mathcal{A}^-_{pq}{}^f \phi_l({}^{m_f}\boldsymbol{\Xi}(\boldsymbol{x}))^{m_f} Q_{lq}(t)\right) \mathrm{d}S \\ &= \sum_{f=0}^{3} {}^m Q_{lq}(t) \mathcal{A}^+_{pq}{}^f |S_f| \int_{\mathcal{E}_2} \phi_k(\boldsymbol{\xi}^f(\boldsymbol{\chi}))\phi_l(\boldsymbol{\xi}^f(\boldsymbol{\chi})) \,\mathrm{d}\boldsymbol{\chi} \\ &+ {}^{m_f} Q_{lq}(t) \mathcal{A}^-_{pq}{}^f |S_f| \int_{\mathcal{E}_2} \phi_k(\boldsymbol{\xi}^f(\boldsymbol{\chi}))\phi_l(\boldsymbol{\xi}^{g_f}(\tilde{\boldsymbol{\chi}}^{h_f}(\boldsymbol{\chi}))) \,\mathrm{d}\boldsymbol{\chi}, \end{aligned} \tag{3.20}$$

where $|S_f| = \left\|(\boldsymbol{x}^{c_f(1)} - \boldsymbol{x}^{c_f(0)}) \times (\boldsymbol{x}^{c_f(2)} - \boldsymbol{x}^{c_f(0)})\right\|$, that is, $|S_f|$ is equal to twice the area of the f-th face of element m.

3.1.4 Summary of the semi-discrete form

Inspecting (3.14) and (3.20), one observes that all integrals are independent of geometry, and may be precomputed in a computer algebra system. In particular, we identify the mass matrix

$$M_{kl} := \int_{\mathcal{E}_3} \phi_k(\boldsymbol{\xi})\phi_l(\boldsymbol{\xi}) \,\mathrm{d}\boldsymbol{\xi}, \tag{3.21}$$

the stiffness matrices

$$K_{kl}{}^e := \int_{\mathcal{E}_3} \frac{\partial \phi_k(\boldsymbol{\xi})}{\partial \xi_e} \phi_l(\boldsymbol{\xi}) \,\mathrm{d}\boldsymbol{\xi}, \tag{3.22}$$

and the flux matrices

$$F^-_{kl}{}^f := \int_{\mathcal{E}_2} \phi_k(\boldsymbol{\xi}^f(\boldsymbol{\chi}))\phi_l(\boldsymbol{\xi}^f(\boldsymbol{\chi})) \,\mathrm{d}\boldsymbol{\chi}, \tag{3.23}$$

$$F^+_{kl}{}^{fgh} := \int_{\mathcal{E}_2} \phi_k(\boldsymbol{\xi}^f(\boldsymbol{\chi}))\phi_l(\boldsymbol{\xi}^g(\tilde{\boldsymbol{\chi}}^h(\boldsymbol{\chi}))) \,\mathrm{d}\boldsymbol{\chi}. \tag{3.24}$$

In summary, the complete semi-discrete scheme is

$$|J|\frac{\partial Q_{lp}(t)}{\partial t}M_{kl} + \sum_{f=0}^{3} Q_{lq}(t)\mathcal{A}^{+}{}_{pq}{}^{f}|S_f|F^{-}{}_{kl}{}^{f} + \sum_{f=0}^{3} {}^{m_f}Q_{lq}(t)\mathcal{A}^{-}{}_{pq}{}^{f}|S_f|F^{+}{}_{kl}{}^{fg_fh_f} - |J|Q_{lq}(t)\Theta^{-1}_{ed}A_{pq}{}^{d}K_{kl}{}^{e} = |J|E_{pq}Q_{lq}(t)M_{kl}. \quad (3.25)$$

Note that tensor contractions are ubiquitous in the above scheme.

3.2 Discretisation in time

The semi-discrete scheme from (3.25) is a system of ordinary differential equations (ODEs). The ADER approach is used to solve these ODEs [81], instead of the commonly employed Runge-Kutta method.

The general idea is to predict the evolution of the PDE in time by neglecting the influence of the boundaries of an element. Plugging in the space-time prediction into the semi-discrete scheme then corrects the prediction and gives an update rule for the degrees of freedom. The predictor can be constructed in multiple ways, but essentially all constructions give similar errors and convergence rates [55].

3.2.1 Cauchy-Kowalevski procedure

The evolution in time is predicted with a truncated Taylor expansion around a time of expansion t_0, i.e.

$$q_p(t) = \sum_{i=0}^{N} \frac{(t-t_0)^i}{i!}\frac{\partial^i q_p}{\partial t^i}(t_0). \quad (3.26)$$

The time derivatives are computed via a Cauchy-Kowalevski procedure, that is, the system of PDEs is used to replace time derivatives by spatial derivatives. E.g. the first derivative in time is

$$\frac{\partial q_p}{\partial t} = -A_{pq}{}^{d}\frac{\partial q_q}{\partial x_d} + E_{pq}q_q \quad (3.27)$$

Due to the linearity of the PDE, and as partial derivatives may be exchanged, higher order derivatives may be computed in the following way:

$$\frac{\partial^i q_p}{\partial t^i} = \left(-A_{pq}{}^d \frac{\partial}{\partial x_d}\right) \frac{\partial^{i-1} q_q}{\partial t^{i-1}} + E_{pq} \frac{\partial^{i-1} q_q}{\partial t^{i-1}} \tag{3.28}$$

The next step is to find a polynomial representation of the time derivatives. That is, given a polynomial basis expansion for the zeroth derivative, $\partial^0 q_p / \partial t^0 = \mathcal{D}_{lp}{}^0 \phi_l$, we want to recover coefficients $\mathcal{D}_{lp}{}^i$, such that $\partial^i q_p / \partial t^i = \mathcal{D}_{lp}{}^i \phi_l$. To do so, one plugs in the basis expansion of the $(i-1)$-th derivative into (3.28) and recovers the coefficients $\mathcal{D}_{lp}{}^i$ via L^2-projection. That is,

$$\begin{aligned}\mathcal{D}_{lp}{}^i \int_{\mathcal{T}_m} \phi_k \phi_l \,\mathrm{d}\boldsymbol{x} = &- \int_{\mathcal{T}_m} \phi_k A_{pq}{}^d \mathcal{D}_{lq}{}^{(i-1)} \frac{\partial \phi_l}{\partial x_d} \,\mathrm{d}\boldsymbol{x} \\ &+ \int_{\mathcal{T}_m} \phi_k E_{pq} \mathcal{D}_{lq}{}^{(i-1)} \phi_l \,\mathrm{d}\boldsymbol{x}\end{aligned} \tag{3.29}$$

Then we map the computation to the reference element, in the same way as in Section 3.1.2, and obtain

$$\begin{aligned}\mathcal{D}_{lp}{}^i |J| \int_{\mathcal{E}_3} \phi_k \phi_l \,\mathrm{d}\boldsymbol{\xi} = &- |J| \Theta^{-1}_{ed} A_{pq}{}^d \mathcal{D}_{lq}{}^{(i-1)} \int_{\mathcal{E}_3} \phi_k \frac{\partial \phi_l}{\partial \xi_e} \,\mathrm{d}\boldsymbol{\xi} \\ &+ |J| E_{pq} \mathcal{D}_{lq}{}^{(i-1)} \int_{\mathcal{E}_3} \phi_k \phi_l \,\mathrm{d}\boldsymbol{\xi}\end{aligned} \tag{3.30}$$

The integrals on the left-hand side are identified as the mass matrix, and the integrals on the right-hand side are identified as the transposed stiffness matrices, which leads to the following scheme:

$$|J| M_{kl} \mathcal{D}_{lp}{}^i = -|J| \Theta^{-1}_{ed} A_{pq}{}^d \mathcal{D}_{lq}{}^{(i-1)} K_{lk}{}^e + |J| E_{pq} \mathcal{D}_{lq}{}^{(i-1)} M_{kl} \tag{3.31}$$

The solution at time t_0 is given by the time-dependent degrees of freedom $q_p(t_0) = Q_{lp}(t_0)\phi_l$, such that $\mathcal{D}_{lp}{}^0 = Q_{lp}(t_0)$. By projecting the Taylor expansion on the polynomial basis, we may predict the time evolution of the degrees of freedom with

$$Q_{lp}(t) = \sum_{i=0}^{N} \frac{(t-t_0)^i}{i!} \mathcal{D}_{lp}{}^i. \tag{3.32}$$

3.2.2 Discrete update scheme

To construct a fully discrete update scheme, we need to introduce a grid in time. One typically uses a uniform grid with vertices t_n, where $t_n = t_{n-1} + \Delta t$. The time-step Δt is restricted by a Courant-Friedrichs-Lewy (CFL) number, due to numerical stability, and must not exceed a maximum time-step $\Delta t_{\max}$.

The CFL number depends on an element's size as well as the maximum wave-speed [38]. Consequently, the CFL number may be very heterogeneous, especially if adaptive meshes are used. So by allowing elements to have different time-steps, commonly referred to as local time-stepping (LTS), many element updates can be saved [18, 41, 158].

We now formally derive the discrete update scheme. Time points of element m are denoted with ${}^m t_n$ (with non-uniform time-steps ${}^m\Delta t$). The fully discrete form is obtained by integrating the semi-discrete scheme from (3.25) in time. The latter is going to require time integrals of the time-dependent degrees of freedom, which are obtained using the Cauchy-Kowalevski procedure from (3.31) and the Taylor expansion from (3.32). We abbreviate time integrals within a time cell $[{}^m t_n, {}^m t_{n+1}]$ with

$$ {}^m\mathcal{L}_{lp}{}^n(t_1, t_0) := \int_{t_0}^{t_1} {}^m Q_{lp}(t)\,\mathrm{d}t = \sum_{i=0}^{N} \frac{(t_1 - {}^m t_n)^i - (t_0 - {}^m t_n)^i}{(i+1)!}\, \mathcal{D}_{lp}{}^i, \tag{3.33} $$

where it is implied that $\mathcal{D}_{lp}{}^0 = {}^m Q_{lp}({}^m t_n) =: {}^m Q_{lp}{}^n$ is used as initial data in (3.31).

Having different time-steps in each element leads to a non-conforming grid in time, such that integrals over several time cells are required. We therefore also introduce the following notation for global time integrals:

$$ {}^m\mathcal{I}_{lp}(t_1, t_0) := \sum_{n \in {}^m\mathcal{N}(t_1, t_0)} {}^m\mathcal{L}_{lp}{}^n \left(\min(t_1, {}^m t_{n+1}), \max(t_0, {}^m t_n)\right), \tag{3.34} $$

where ${}^m\mathcal{N}(t_1, t_0) := \{n \in \mathbb{N}_0 : ({}^m t_n, {}^m t_{n+1}) \cap (t_0, t_1) \neq \emptyset\}$. Note that open sets are used in the definition of ${}^m\mathcal{N}$. This formulation automatically avoids integrals over sets of measure zero.

Integrating the semi-discrete scheme from (3.25) in time in the interval $[{}^m t_n, {}^m t_{n+1}]$ gives the update scheme for the discrete degrees of freedom ${}^m Q_{lp}{}^{n+1}$ (we drop the superscript m for better readability):

$$\begin{aligned} |J| \left(Q_{lp}{}^{n+1} - Q_{lp}{}^{n} \right) M_{kl} + \sum_{f=0}^{3} \mathcal{I}_{lq}(t_{n+1}, t_n)\, \mathcal{A}^{+}{}_{pq}{}^{f} |S_f| F^{-}{}_{kl}{}^{f} \\ + \sum_{f=0}^{3} {}^{m_f}\mathcal{I}_{lq}(t_{n+1}, t_n)\, \mathcal{A}^{-}{}_{pq}{}^{f} |S_f| F^{+}{}_{kl}{}^{f g_f h_f} \\ - |J| \mathcal{I}_{lq}(t_{n+1}, t_n) \Theta^{-1}_{ed} A_{pq}{}^{d} K_{kl}{}^{e} = |J| E_{pq} \mathcal{I}_{lq}(t_{n+1}, t_n) M_{kl}. \end{aligned} \tag{3.35}$$

Equation (3.35) is the fully discrete ADER-DG scheme. Further tweaks to the scheme for an efficient implementation are discussed in Chapter 7.

3.3 Non-linear numerical flux

So far in this chapter, we have assumed that the numerical flux is a linear combination of the one-sided limits of an interface, cf. (3.4), which is not true for non-linear boundary conditions and dynamic rupture [123, 124].

In this section, we assume that the numerical flux is given by

$$\left(n_d A_{pq}{}^{d} q_q \right)^{*} = n_d A_{pq}{}^{d} \widetilde{q}_q \left(\lim_{\epsilon \to 0} \boldsymbol{q}(\boldsymbol{x} - \epsilon \boldsymbol{n}, t), \lim_{\epsilon \to 0} \boldsymbol{q}(\boldsymbol{x} + \epsilon \boldsymbol{n}, t) \right), \tag{3.36}$$

where $\widetilde{q}_q$ are non-linear functions that depend on both sides of an interface. The parameterisation of the surface integrals flux is essentially the same as in Section 3.1.3. Using the definition

$$\begin{aligned} q^{-}{}_{p}{}^{f}(\boldsymbol{\chi}, t) &:= \phi_l(\boldsymbol{\xi}^f(\boldsymbol{\chi}))\, {}^m Q_{lp}(t), \\ q^{+}{}_{p}{}^{f}(\boldsymbol{\chi}, t) &:= \phi_l(\boldsymbol{\xi}^{g_f}(\tilde{\boldsymbol{\chi}}^{h_f}(\boldsymbol{\chi})))\, {}^{m_f} Q_{lp}(t), \end{aligned} \tag{3.37}$$

we find that the time-integrated surface integral of face f is given by

$$\begin{aligned} \mathcal{F}_{kp}{}^{f} &:= \int_{t_n}^{t_{n+1}} \int_{\partial(\mathcal{T}_m)_f} \phi_k({}^m \boldsymbol{\Xi}(\boldsymbol{x}))(n_d A_{pq}{}^{d} q_q)^{*} \,\mathrm{d}S \,\mathrm{d}t \\ &= n_d A_{pq}{}^{d} |S_f| \int_{t_n}^{t_{n+1}} \int_{\mathcal{E}_2} \phi_k(\boldsymbol{\xi}^f(\boldsymbol{\chi})) \widetilde{q}_q \left(\boldsymbol{q}^{-f}(\boldsymbol{\chi}, t), \boldsymbol{q}^{+f}(\boldsymbol{\chi}, t) \right) \mathrm{d}\boldsymbol{\chi} \,\mathrm{d}t. \end{aligned} \tag{3.38}$$

Following Pelties et al. [123], the above integrals are approximated using an appropriate quadrature rule on the reference triangle, $(\beta_i, \boldsymbol{\chi}_i)_{i=1,\dots,N^s}$, and an appropriate one-dimensional quadrature rule for time, $(\gamma_z, \tau_z)_{z=1,\dots,N^t}$:

$$\mathcal{F}_{kp}{}^{f} \approx n_d A_{pq}{}^{d} |S_f| \sum_{i=1}^{N^s} \sum_{z=1}^{N^t} \beta_i \gamma_z \phi_k(\boldsymbol{\xi}^f(\boldsymbol{\chi}_i)) \widetilde{q}_q \left(\boldsymbol{q}^{-f}(\boldsymbol{\chi}_i, \tau_z), \boldsymbol{q}^{+f}(\boldsymbol{\chi}_i, \tau_z) \right). \tag{3.39}$$

In comparison to the time-integrated surface integrals for linear numerical fluxes, non-linear numerical fluxes are expensive to compute – even if we disregard the evaluation of $\widetilde{q}_q$. In particular, the numerical flux has to be computed for several points in time, which includes several evaluations of the Taylor polynomial in time using (3.32).

3.4 Point sources

Kinematic rupture models add a delta distribution to the right-hand side of the wave equations as shown in Section 2.4.1. These terms have the general form $s_{pc}(t)\delta(\boldsymbol{x} - \boldsymbol{x}_c)$, where the locations of the point sources are given by $\boldsymbol{x}_c$ and we allow a separate source time function s_{pc} for every quantity and point source. Following Käser et al. [83], the source terms are implemented in the ADER-DG scheme by adding

$${}^{m}\mathcal{S}_{kp}{}^{n} := \sum_{c} \begin{cases} \phi_k({}^{m}\boldsymbol{\Xi}(\boldsymbol{x}_c)) \int_{t_n}^{t_{n+1}} s_{pc}(t)\,\mathrm{d}t & \text{if} \quad \mathrm{MapPoint}(\boldsymbol{x}_c) = m, \\ 0 & \text{else.} \end{cases} \tag{3.40}$$

to the right-hand side of (3.35). The point source is only evaluated in elements that contain the point $\boldsymbol{x}_c$. It may happen that a point is part of several elements, e.g. if it resides on a face or a vertex. In this case, the point source is mapped to a unique element. This behaviour is encapsulated in the MapPoint function, which returns a unique element number m_c with the property $\boldsymbol{x}_c \subset \mathcal{T}_{m_c}$.

CHAPTER

4

Numerical flux and boundary conditions

The numerical flux lies at the heart of every discontinuous Galerkin scheme. Its role is to ensure consistency and stability [68]. It is imperative to choose an appropriate numerical flux for convergence.

Several common choices for the numerical flux exist, the simplest arguably being the Lax-Friedrichs flux, where one only requires an estimate of the maximum eigenvalue of the directional Jacobian of a system of PDEs. More accurate and less dissipative fluxes may be obtained by considering the underlying physics. Here, a common approach is to solve a plane-wave Riemann problem at every point of an interface [100, 151], which is derived in detail in the remainder of this chapter.

The motivation for a separate chapter for the numerical flux is that current versions of SeisSol use a different numerical flux than described in the original papers [39, 80, 81, 126]. The original numerical flux is "one-sided", which means that only material parameters from one side are taken into account in the numerical flux, even when different material parameters are present at a heterogeneous material interface. In particular, for simulations coupling elastic and acoustic media, the one-sided flux [80] has been criticised to lead to an inconsistent numerical scheme [162].

The methodology to derive Riemann solvers for linear PDEs with heterogeneous material parameters is not new (e.g. [68, 100, 162]). However, in SeisSol related publications, the concrete *two-sided* numerical fluxes have only been mentioned briefly for the elastic wave equation in [122,

Appendix A] and [16, Ch. 2.9]. Therefore, a comprehensive overview of the two-sided numerical flux is given in Section 4.2, including the elastic wave equation, viscoelastic wave equation, acoustic wave equation, and the coupling of the acoustic and elastic wave equation.

Moreover, dynamic rupture, free surface, and absorbing boundary conditions are discussed in Section 4.3. The derivation of dynamic rupture is inspired by [42], a finite difference method in which the fault boundary conditions are imposed weakly. As both [42] and [122–124] essentially consider characteristics propagating to and from the fault, the two approaches are closely related, which becomes evident from the derivation in Section 4.3.1.

4.1 Plane-wave Riemann problem

We recall from the last chapter that we are looking for a numerical flux which approximates $\left(n_d A_{pq}{}^d q_q\right)^*$, i.e. the solution on an interface with normal $\boldsymbol{n}$. A commonly used method is to define the numerical flux as the solution of a local initial value problem (IVP), which should capture most of the underlying physics.

Simplifying assumptions are made. First, as only short-term interactions are of interest, one only considers initial data of elements adjacent to the interface. Second, the major source of interaction stems from the possible discontinuity along the interface. Third, source terms are neglected [151, Sec. 19.4.1]. Therefore, one considers the following IVP with a plane-wave initial condition:[1]

$$\begin{aligned} \frac{\partial q_p}{\partial \tau} + A_{pq}{}^d \frac{\partial q_q}{\partial \nu_d} &= 0, \\ q_p(\boldsymbol{\nu}, 0) &= \begin{cases} q_p^-(\boldsymbol{x}, t) & \text{if} \quad \boldsymbol{n} \cdot (\boldsymbol{\nu} - \boldsymbol{x}) < 0, \\ q_p^+(\boldsymbol{x}, t) & \text{else,} \end{cases} \end{aligned} \tag{4.1}$$

where $q_p^{\pm}(\boldsymbol{x}, t) = \lim_{\epsilon \to 0} q_p(\boldsymbol{x} \pm \epsilon \boldsymbol{n}, t)$. As the initial condition varies only in normal direction, we parameterise $\boldsymbol{\nu} = \boldsymbol{x} + \beta \boldsymbol{n} + \gamma_1 \boldsymbol{t}_1 + \gamma_2 \boldsymbol{t}_2$, for an

[1] Actually, one has to consider the generalised Riemann problem with polynomial initial data for high-order accuracy [151, Ch. 19]. However, in the linear case, one only requires the solution to the standard Riemann problem, therefore we skip the discussion of generalised Riemann problems.

orthonormal basis $(\boldsymbol{n}, \boldsymbol{t}_1, \boldsymbol{t}_2)$. We find the solution to (4.1) by solving the following one-dimensional IVP:

$$\begin{aligned} \frac{\partial q_p}{\partial \tau} + n_d A_{pq}{}^d \frac{\partial q_q}{\partial \beta} &= 0, \\ q_p(\beta, 0) &= \begin{cases} q_p^- & \text{if} \quad \beta < 0, \\ q_p^+ & \text{else}, \end{cases} \end{aligned} \tag{4.2}$$

For the numerical flux we evaluate the solution $q_p(\beta, \tau)$ to (4.2) on the interface at the first instant in time [151, Sec. 19.4.1]. Hence, the numerical flux is defined as following:

$$\left(n_d A_{pq}{}^d q_q\right)^* = n_d A_{pq}{}^d \lim_{\tau \to 0} q_q(0, \tau) \tag{4.3}$$

The eigendecomposition of $n_d A_{pq}{}^d$ is required to solve IVP (4.2). As the latter operator depends on $\boldsymbol{n}$, it may be simpler to make use of a rotational invariance property, which takes the form

$$n_d A_{pq}{}^d = T_{pm}(\boldsymbol{n}) A_{mn}{}^1 T_{nq}^{-1}(\boldsymbol{n}) \tag{4.4}$$

and exists for a multitude of rheological models [39, 81, 125, 126]. (The rotational invariance property for the elastic wave equation is discussed in detail in Section 5.1.) Using (4.4) one may rewrite (4.2) as

$$\begin{aligned} \frac{\partial w_p}{\partial \tau} + A_{pq}{}^1 \frac{\partial w_q}{\partial \beta} &= 0, \\ w_p(\beta, 0) = \mathring{w}_p(\beta) &= \begin{cases} w_p^- = T_{pm}^{-1} q_m^- & \text{if} \quad \beta < 0, \\ w_p^+ = T_{pm}^{-1} q_m^+ & \text{else.} \end{cases} \end{aligned} \tag{4.5}$$

Solving the above one-dimensional Riemann problem only requires the eigendecomposition of $A_{pq}{}^1$ which is likely simpler to obtain than the eigendecomposition of $n_d A_{pq}{}^d$.

4.1.1 Riemann problems in homogeneous media

We investigate the solution structure of (4.5), where we assume that a rotational invariance property is available and that the initial condition was rotated to an interface-aligned coordinate system. The superscript for A is dropped in the following, so we abbreviate $A_{pq}{}^1$ with A_{pq}.

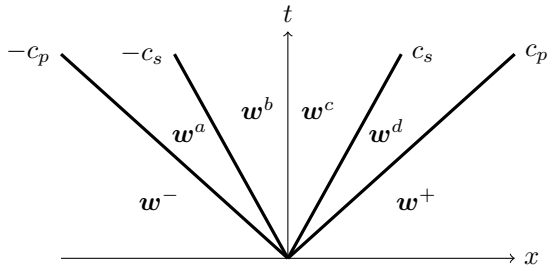


Figure 6: Solution structure of the one-dimensional Riemann problem for the elastic wave equation. The characteristics curves (lines) in the space-time plane show propagating discontinuities with speed $\pm c_s$ or $\pm c_p$. In between lines, the solution takes constant states.

In the homogeneous case, where A is space-invariant, the IVP may be diagonalized with the eigendecomposition of $A = R\Lambda R^{-1}$. Substituting $w = Rv$ we obtain

$$\begin{aligned} &\frac{\partial v_p}{\partial \tau} + \Lambda_{pq} \frac{\partial v_q}{\partial \beta} = 0, \\ &v_p(\beta, 0) = \mathring{v}_p(\beta) = \begin{cases} v_p^- & \text{if} \quad \beta < 0, \\ v_p^+ & \text{else,} \end{cases} \end{aligned} \tag{4.6}$$

with $v_p^\pm = R_{pm}^{-1} w_m^\pm$. The system now consists of several decoupled advection equations with the general solution[2]

$$v_p(\beta, \tau) = \mathring{v}_p(\beta - \lambda_p \tau). \tag{4.7}$$

That is, the initial condition is shifted and moves with speed λ_p. As the initial condition consists of constant states separated by discontinuities, the solution consists of a finite number of states. A typical plot of the solution structure in the space-time plane is shown in Figure 6.

[2] The term "solution" must not be interpreted in a strong sense, as the strong derivative at $\mathring{v}_p(0)$ does clearly not exist. However, one my define a weak derivative for $\mathring{v}_p(0)$, in which the derivative at the discontinuity makes sense, and consequently a "weak solution". The weak derivatives are element of a Sobolev space. As a consequence, the value $\mathring{v}_p(0)$ becomes irrelevant, as functions that differ on a set of measure zero are equivalent [15].

For the numerical flux we require the solution at $\beta = 0$ and at the first instant in time:

$$\lim_{\tau\to 0^+} v_p(0,\tau) = \lim_{\tau\to 0^+} \overset{\circ}{v}_p(-\lambda_p\tau) = \begin{cases} v_p^- & \text{if} \quad \lambda_p > 0, \\ v_p^+ & \text{if} \quad \lambda_p < 0. \end{cases} \tag{4.8}$$

(The case $\lambda_p = 0$ is left out on purpose.) Multiplying the above equation with R and inserting $v^\pm$ yields

$$\lim_{\tau\to 0^+} w_p(0,\tau) = \sum_{\lambda_n>0} R_{pn} \sum_m R^{-1}_{nm} w^-_m + \sum_{\lambda_n<0} R_{pn} \sum_m R^{-1}_{nm} w^+_m. \tag{4.9}$$

It is illustrative to rewrite the above equation using the indicator matrix χ^+ (χ^-), which has a 1 on the p-th diagonal entry if λ_p is positive (negative) and 0 otherwise. We obtain

$$\lim_{\tau\to 0^+} \boldsymbol{w}(0,\tau) = R\chi^+R^{-1}\boldsymbol{w}^- + R\chi^-R^{-1}\boldsymbol{w}^+ \tag{4.10}$$

The numerical flux is obtained by multiplication with A:

$$\begin{aligned}(A\boldsymbol{w})^* = A \lim_{\tau\to 0^+} \boldsymbol{w}(0,\tau) &= AR\chi^+R^{-1}\boldsymbol{w}^- + AR\chi^-R^{-1}\boldsymbol{w}^+ \\ &= R\Lambda^+R^{-1}\boldsymbol{w}^- + R\Lambda^-R^{-1}\boldsymbol{w}^+, \end{aligned} \tag{4.11}$$

where $\Lambda^\pm = \Lambda\chi^\pm$. In retrospect, we see that zero eigenvalues do not influence the numerical flux: The zero-indicator matrix χ^0 multiplied with Λ simply yields the zero matrix.

Before we continue to the next section, an alternative way to derive the same numerical flux is introduced. Any shock that appears in a conservation law must obey the Rankine-Hugeniot condition [100, 151], which states that every jump $[\![\boldsymbol{w}]\!]$ must obey the following condition

$$A[\![\boldsymbol{w}]\!] = s[\![\boldsymbol{w}]\!], \tag{4.12}$$

where the scalar s is called the shock speed. For linear PDEs, this is an eigenproblem and hence the jump must be an eigenvector of A and the shock speed must be an eigenvalue of A. E.g. for the elastic wave equation one obtains from the eigendecomposition that the solution to the Riemann problem must consist of two left-going waves with speeds $-c_p$ and $-c_s$

and two right-going waves with speeds c_s and c_p, cf. Figure 6. Thus, the following Rankine-Hugeniot conditions need to be satisfied:

$$\begin{aligned} A(\boldsymbol{w}^a - \boldsymbol{w}^-) &= -c_p(\boldsymbol{w}^a - \boldsymbol{w}^-), \\ A(\boldsymbol{w}^b - \boldsymbol{w}^a) &= -c_s(\boldsymbol{w}^b - \boldsymbol{w}^a), \\ A(\boldsymbol{w}^c - \boldsymbol{w}^b) &= \boldsymbol{0}, \\ A(\boldsymbol{w}^d - \boldsymbol{w}^c) &= c_s(\boldsymbol{w}^d - \boldsymbol{w}^c), \\ A(\boldsymbol{w}^+ - \boldsymbol{w}^d) &= c_p(\boldsymbol{w}^+ - \boldsymbol{w}^d), \end{aligned} \tag{4.13}$$

where each jump $[\![\boldsymbol{w}]\!]$ must lie in the respective eigenspace, i.e.

$$\begin{aligned} \boldsymbol{w}^a - \boldsymbol{w}^- &= \alpha_1 \boldsymbol{r}_1, \\ \boldsymbol{w}^b - \boldsymbol{w}^a &= \alpha_2 \boldsymbol{r}_2 + \alpha_3 \boldsymbol{r}_3, \\ \boldsymbol{w}^c - \boldsymbol{w}^b &= \alpha_4 \boldsymbol{r}_4 + \alpha_5 \boldsymbol{r}_5 + \alpha_6 \boldsymbol{r}_6, \\ \boldsymbol{w}^d - \boldsymbol{w}^c &= \alpha_7 \boldsymbol{r}_7 + \alpha_8 \boldsymbol{r}_8, \\ \boldsymbol{w}^+ - \boldsymbol{w}^d &= \alpha_9 \boldsymbol{r}_9. \end{aligned} \tag{4.14}$$

One may derive that

$$\begin{aligned} \boldsymbol{w}^b &= \boldsymbol{w}^- + \sum_{i=1}^{3} \alpha_i \boldsymbol{r}_i = \boldsymbol{w}^- + R\chi^- \boldsymbol{\alpha} \\ \boldsymbol{w}^c &= \boldsymbol{w}^+ - \sum_{i=7}^{9} \alpha_i \boldsymbol{r}_i = \boldsymbol{w}^+ - R\chi^+ \boldsymbol{\alpha} \end{aligned} \tag{4.15}$$

It is left as an exercise to the reader to check that $A\boldsymbol{w}^c = A\boldsymbol{w}^b = (A\boldsymbol{w})^*$ with $\boldsymbol{\alpha} = R^{-1}(\boldsymbol{w}^+ - \boldsymbol{w}^-)$ and $(A\boldsymbol{w})^*$ as in (4.11).

4.1.2 Riemann problems in heterogeneous media

In this section we consider Riemann problems with piecewise-constant material parameters. That is, we look for solutions of the IVP

$$\begin{aligned} &\frac{\partial w_p}{\partial \tau} + A_{pq}(\beta)\frac{\partial w_q}{\partial \beta} = 0, \\ &A(\beta) = \begin{cases} A^- & \text{if} \quad \beta < 0, \\ A^+ & \text{else}, \end{cases} \\ &w_p(\beta, 0) = \mathring{w}_p(\beta) = \begin{cases} w_p^- & \text{if} \quad \beta < 0, \\ w_p^+ & \text{else}. \end{cases} \end{aligned} \tag{4.16}$$

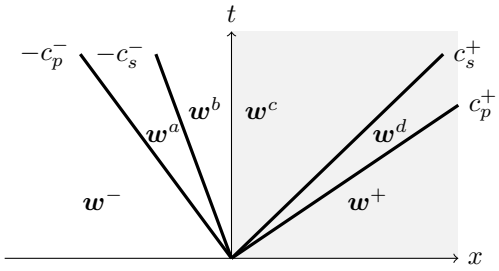


Figure 7: Solution structure of the one-dimensional Riemann problem for the elastic wave equation in heterogeneous media. The material parameters are constant but differ in the left and right half-plane, leading to to different wave velocities. A similar sketch is shown in [162].

For $\beta < 0$ and $\beta > 0$ we recognise the homogeneous media case from Section 4.1.1, and infer that the Riemann solution has to consist of left- and right-going waves, whose velocity depends on the medium. Moreover, the Rankine-Hugeniot conditions have to hold for jumps in the left half-space and right half-space [68, e.g.]. However, we cannot treat jumps across the interface at $x = 0$ with the Rankine-Hugeniot conditions, as the flux is generally not conserved in Equation (4.16). Here, one may introduce the proper physical interface conditions [52], e.g. for the elastic wave equation one would enforce continuity of traction and continuity of particle velocity.

For example, the solution structure of the elastic wave equation in heterogeneous media is depicted in Figure 7, and we obtain the following equations from the Rankine-Hugeniot condition:

$$\begin{aligned} A^-(\boldsymbol{w}^a - \boldsymbol{w}^-) &= -c_p^-(\boldsymbol{w}^a - \boldsymbol{w}^-), \\ A^-(\boldsymbol{w}^b - \boldsymbol{w}^a) &= -c_s^-(\boldsymbol{w}^b - \boldsymbol{w}^a), \\ A^+(\boldsymbol{w}^d - \boldsymbol{w}^c) &= c_s^+(\boldsymbol{w}^d - \boldsymbol{w}^c), \\ A^+(\boldsymbol{w}^+ - \boldsymbol{w}^d) &= c_p^+(\boldsymbol{w}^+ - \boldsymbol{w}^d), \end{aligned} \quad \text{and} \quad \begin{aligned} \boldsymbol{w}^a - \boldsymbol{w}^- &= \alpha_1 \boldsymbol{r}_1^-, \\ \boldsymbol{w}^b - \boldsymbol{w}^a &= \alpha_2 \boldsymbol{r}_2^- + \alpha_3 \boldsymbol{r}_3^-, \\ \boldsymbol{w}^d - \boldsymbol{w}^c &= \alpha_7 \boldsymbol{r}_7^+ + \alpha_8 \boldsymbol{r}_8^+, \\ \boldsymbol{w}^+ - \boldsymbol{w}^d &= \alpha_9 \boldsymbol{r}_9^+. \end{aligned} \tag{4.17}$$

Together with the physical boundary condition, which relate $\boldsymbol{w}^b$ and $\boldsymbol{w}^c$, this forms a linear system of equations, which may be used to derive a numerical flux in the same way as in the homogeneous case.

4.2 Numerical fluxes for various rheological models

Having laid out the basics of physically motivated numerical fluxes, we present a few examples for various media. First of all, we derive the numerical fluxes for the elastic wave equation in heterogeneous media, as well as the numerical fluxes for viscoelastic media. Then, numerical fluxes for coupling the acoustic wave equation and elastic wave equation are derived. The latter is of interest for investigating coupled earthquake-tsunami events [103, 104].

Note that for all covered problems a rotational invariance property exists, hence we look for solutions of the standard problem (4.16).

4.2.1 Elastic wave equation

The space-dependent coefficient matrix A is given by [39]

$$A = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & -(\lambda+2\mu) & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -\lambda & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -\lambda & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\mu & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\mu \\ -\frac{1}{\rho} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -\frac{1}{\rho} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -\frac{1}{\rho} & 0 & 0 & 0 \end{pmatrix} \tag{4.18}$$

for the vector of quantities $(\sigma_{xx}, \sigma_{yy}, \sigma_{zz}, \sigma_{xy}, \sigma_{yz}, \sigma_{xz}, u_1, u_2, u_3)$. The eigenvalues are $(-c_p, -c_s, -c_s, 0, 0, 0, c_s, c_s, c_p)$, where the P-wave speed c_p and the S-wave speed c_s are related to the Lamé parameters and the density by

$$c_p = \sqrt{\frac{\lambda+2\mu}{\rho}} \quad \text{and} \quad c_s = \sqrt{\frac{\mu}{\rho}}. \tag{4.19}$$

The corresponding eigenvectors are [39]

$$R = \begin{pmatrix} \lambda+2\mu & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \lambda+2\mu \\ \lambda & 0 & 0 & 0 & 1 & 0 & 0 & 0 & \lambda \\ \lambda & 0 & 0 & 0 & 0 & 1 & 0 & 0 & \lambda \\ 0 & \mu & 0 & 0 & 0 & 0 & 0 & \mu & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \mu & 0 & 0 & 0 & \mu & 0 & 0 \\ c_p & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -c_p \\ 0 & c_s & 0 & 0 & 0 & 0 & 0 & -c_s & 0 \\ 0 & 0 & c_s & 0 & 0 & 0 & -c_s & 0 & 0 \end{pmatrix}. \tag{4.20}$$

We infer from (4.17) that

$$\begin{aligned} \boldsymbol{w}^b - \boldsymbol{w}^- &= \alpha_1 \boldsymbol{r}_1^- + \alpha_2 \boldsymbol{r}_2^- + \alpha_3 \boldsymbol{r}_3^-, \\ \boldsymbol{w}^+ - \boldsymbol{w}^c &= \alpha_7 \boldsymbol{r}_7^+ + \alpha_8 \boldsymbol{r}_8^+ + \alpha_9 \boldsymbol{r}_9^+, \end{aligned} \tag{4.21}$$

where $\boldsymbol{r}_i^{\pm}$ are columns of R w.r.t. the correct domain ('$-$' for $\beta < 0$ and '$+$' for $\beta > 0$). The physical boundary condition is continuity of traction and continuity of particle velocity. The traction vector is in general given by $t_i = \sigma_{ij} n_j$, and here by $(\sigma_{xx}, \sigma_{xy}, \sigma_{xz})$. Hence, the physical boundary conditions translate to

$$\begin{aligned} \sigma_{xx}^b &= \sigma_{xx}^c, & u_1^b &= u_1^c, \\ \sigma_{xy}^b &= \sigma_{xy}^c, & u_2^b &= u_2^c, \\ \sigma_{xz}^b &= \sigma_{xz}^c, & u_3^b &= u_3^c. \end{aligned} \tag{4.22}$$

The number of unknowns matches the number of physical boundary conditions, such that from (4.21) and (4.22) we get a linear system of equations with a unique solution:

$$\begin{pmatrix} \lambda^- + 2\mu^- & 0 & 0 & 0 & 0 & \lambda^+ + 2\mu^+ \\ 0 & \mu^- & 0 & 0 & \mu^+ & 0 \\ 0 & 0 & \mu^- & \mu^+ & 0 & 0 \\ c_p^- & 0 & 0 & 0 & 0 & -c_p^+ \\ 0 & c_s^- & 0 & 0 & -c_s^+ & 0 \\ 0 & 0 & c_s^- & -c_s^+ & 0 & 0 \end{pmatrix} \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_7 \\ \alpha_8 \\ \alpha_9 \end{pmatrix} = \begin{pmatrix} \sigma_{xx}^+ - \sigma_{xx}^- \\ \sigma_{xy}^+ - \sigma_{xy}^- \\ \sigma_{xz}^+ - \sigma_{xz}^- \\ u_1^+ - u_1^- \\ u_2^+ - u_2^- \\ u_3^+ - u_3^- \end{pmatrix} \tag{4.23}$$

The system becomes trivial to solve once a favourable permutation is chosen, as only pairs of variables are coupled in the system of equations:

$$\begin{pmatrix} \lambda^- + 2\mu^- & \lambda^+ + 2\mu^+ & 0 & 0 & 0 & 0 \\ c_p^- & -c_p^+ & 0 & 0 & 0 & 0 \\ 0 & 0 & \mu^- & \mu^+ & 0 & 0 \\ 0 & 0 & c_s^- & -c_s^+ & 0 & 0 \\ 0 & 0 & 0 & 0 & \mu^- & \mu^+ \\ 0 & 0 & 0 & 0 & c_s^- & -c_s^+ \end{pmatrix} \begin{pmatrix} \alpha_1 \\ \alpha_9 \\ \alpha_2 \\ \alpha_8 \\ \alpha_3 \\ \alpha_7 \end{pmatrix} = \begin{pmatrix} \sigma_{xx}^+ - \sigma_{xx}^- \\ u_1^+ - u_1^- \\ \sigma_{xy}^+ - \sigma_{xy}^- \\ u_2^+ - u_2^- \\ \sigma_{xz}^+ - \sigma_{xz}^- \\ u_3^+ - u_3^- \end{pmatrix} \tag{4.24}$$

The numerical flux is then obtained with $A^- \boldsymbol{w}^b$ for the minus side and $A^+ \boldsymbol{w}^c$ for the plus side.

We can also state the above procedure in a more generic way: First we define the matrix

$$R^{\pm} = \begin{pmatrix} \boldsymbol{r}_1^- & \boldsymbol{r}_2^- & \boldsymbol{r}_3^- & \boldsymbol{e}_5 & \boldsymbol{e}_2 & \boldsymbol{e}_3 & \boldsymbol{r}_7^+ & \boldsymbol{r}_8^+ & \boldsymbol{r}_9^+ \end{pmatrix}, \tag{4.25}$$

which is nothing else than R but with eigenvectors taken from the respective sides. (The vectors $\boldsymbol{e}_5, \boldsymbol{e}_2, \boldsymbol{e}_3$ span the null-space of R and are identical to columns 4–6 of R.) The alphas are then given by

$$\boldsymbol{\alpha} = (R^{\pm})^{-1}(\boldsymbol{w}^+ - \boldsymbol{w}^-) \tag{4.26}$$

and the numerical fluxes are

$$\begin{aligned} A^-\boldsymbol{w}^b &= A^- R^{\pm}\chi^+(R^{\pm})^{-1}\boldsymbol{w}^- + A^- R^{\pm}\chi^-(R^{\pm})^{-1}\boldsymbol{w}^+, \\ A^+\boldsymbol{w}^c &= A^+ R^{\pm}\chi^+(R^{\pm})^{-1}\boldsymbol{w}^- + A^+ R^{\pm}\chi^-(R^{\pm})^{-1}\boldsymbol{w}^+. \end{aligned} \tag{4.27}$$

4.2.2 Viscoelastic wave equation

The coefficient matrix for the viscoelastic wave equation can be partitioned in the following way [81]:

$$\breve{A} := \begin{pmatrix} A & 0_{9\times(6L)} \\ \breve{B} & 0_{(6L)\times(6L)} \end{pmatrix}, \tag{4.28}$$

where A is the coefficient matrix of the elastic wave equation, i.e. given by (4.18), L is the number of relaxation mechanisms, and $\breve{B}$ is a $(6L) \times 9$ matrix.

The eigenproblem of $\breve{A}$ reduces to

$$\begin{pmatrix} A & 0 \\ \breve{B} & 0 \end{pmatrix} \begin{pmatrix} \boldsymbol{w} \\ \boldsymbol{w}^{\text{ane}} \end{pmatrix} = \lambda \begin{pmatrix} \boldsymbol{w} \\ \boldsymbol{w}^{\text{ane}} \end{pmatrix} \quad \Leftrightarrow \quad \begin{aligned} A\boldsymbol{w} &= \lambda \boldsymbol{w}, \\ \breve{B}\boldsymbol{w} &= \lambda \boldsymbol{w}^{\text{ane}}. \end{aligned} \tag{4.29}$$

So as in the elastic case, we have 6 non-zero eigenvalues and the eigenvectors corresponding to non-zero eigenvalues are given by

$$\breve{\boldsymbol{r}}_i = \begin{pmatrix} \boldsymbol{r}_i \\ \lambda_i^{-1}\breve{B}\boldsymbol{r}_i \end{pmatrix}, \tag{4.30}$$

where the pairs $(\lambda_i, \boldsymbol{r}_i)$ are identical to those in Section 4.2.1. Therefore, the eigendecomposition is $\breve{A} = \breve{R}\breve{\Lambda}\breve{R}^{-1}$, where

$$\breve{R} = \begin{pmatrix} R & 0 \\ \Lambda^\dagger \breve{B} R & I \end{pmatrix}, \quad \breve{\Lambda} = \begin{pmatrix} \Lambda & 0 \\ 0 & 0 \end{pmatrix}, \quad \breve{R}^{-1} = \begin{pmatrix} R^{-1} & 0 \\ -\Lambda^\dagger \breve{B} & I \end{pmatrix}. \tag{4.31}$$

We take the pseudo-inverse of Λ with the dagger symbol as Λ has zeros on the diagonal.

Following the generic procedure from Section 4.2.1 we obtain the numerical fluxes

$$\begin{aligned} \breve{A}^- \breve{\boldsymbol{w}}^b &= \breve{A}^- \breve{R}^\pm \breve{\chi}^+ (\breve{R}^\pm)^{-1} \breve{\boldsymbol{w}}^- + \breve{A}^- \breve{R}^\pm \breve{\chi}^- (\breve{R}^\pm)^{-1} \breve{\boldsymbol{w}}^+, \\ \breve{A}^+ \breve{\boldsymbol{w}}^c &= \breve{A}^+ \breve{R}^\pm \breve{\chi}^+ (\breve{R}^\pm)^{-1} \breve{\boldsymbol{w}}^- + \breve{A}^+ \breve{R}^\pm \breve{\chi}^- (\breve{R}^\pm)^{-1} \breve{\boldsymbol{w}}^+, \end{aligned} \tag{4.32}$$

where $\breve{\chi}^+ = \mathrm{diag}(\chi^+, 0_{(6L)\times(6L)})$ and $\breve{\chi}^- = \mathrm{diag}(\chi^-, 0_{(6L)\times(6L)})$.

We explicitly compute

$$\breve{A}^- \breve{R}^\pm \breve{\chi}^+ (\breve{R}^\pm)^{-1} = \begin{pmatrix} A^- R^\pm \chi^+ (R^\pm)^{-1} & 0 \\ \breve{B}^- R^\pm \chi^+ (R^\pm)^{-1} & 0 \end{pmatrix}. \tag{4.33}$$

The other three matrices in (4.32) have the same structure. These computations show that the numerical flux depends only on the elastic quantities.

4.2.3 Coupling of acoustic and elastic waves

The acoustic wave equation can be seen as a special case of the elastic wave equation by setting the shear modulus to zero [80]. Indeed, inserting $\mu = 0$ in (2.8) gives

$$\begin{aligned} \frac{\partial}{\partial t}\sigma_{ij} - \lambda \delta_{ij} \frac{\partial}{\partial x_k} u_k &= 0, \\ \rho \frac{\partial}{\partial t} u_i - \frac{\partial}{\partial x_j} \sigma_{ji} &= 0. \end{aligned} \tag{4.34}$$

The equations for the bulk stresses are identical and the shear stresses have to be constant in time. Hence, identifying pressure with bulk stresses, i.e.

$p := \sigma_{xx} = \sigma_{yy} = \sigma_{zz}$, initialising shear stresses to zero, i.e. $\sigma_{xy} = \sigma_{yz} = \sigma_{xz} = 0$, yields the wave equations for acoustics [100]:

$$\begin{aligned} \frac{\partial}{\partial t} p - \lambda \frac{\partial}{\partial x_k} u_k = 0, \\ \rho \frac{\partial}{\partial t} u_i - \frac{\partial}{\partial x_i} p = 0. \end{aligned} \tag{4.35}$$

In our standard Riemann problem, we use the same quantities as for the elastic wave equation with the coefficient matrix A^{acoustic}, given by

$$A^{\text{acoustic}} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & -\lambda & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -\lambda & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -\lambda & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -\frac{1}{\rho} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}. \tag{4.36}$$

The eigenvalues of A^{acoustic} are $(-c_p, 0, 0, 0, 0, 0, 0, 0, c_p)$ and corresponding eigenvectors are

$$R^{\text{acoustic}} = \begin{pmatrix} \lambda & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \lambda \\ \lambda & 0 & 0 & 0 & 1 & 0 & 0 & 0 & \lambda \\ \lambda & 0 & 0 & 0 & 0 & 1 & 0 & 0 & \lambda \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ c_p & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -c_p \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}. \tag{4.37}$$

The wave speed c_p is defined by (4.19) with $\mu = 0$. We also note that the first and last column of R^{acoustic} are identical with R when $\mu = 0$, hence we use $\boldsymbol{r}_1$ and $\boldsymbol{r}_9$ to refer to both matrices.

At an elastic-acoustic interface the pressure and the velocity normal to the interface must be continuous [141]. Here, this means that u must be continuous, whereas v and w need not, and that $\sigma_{xx}, \sigma_{xy}, \sigma_{xz}$ must be continuous. This ensures that pressure is continuous and that shear stresses are zero at an elastic-acoustic interface. Figure 8 shows the solution structure of all possible configurations, for which we derive the solution in the remainder of this section.

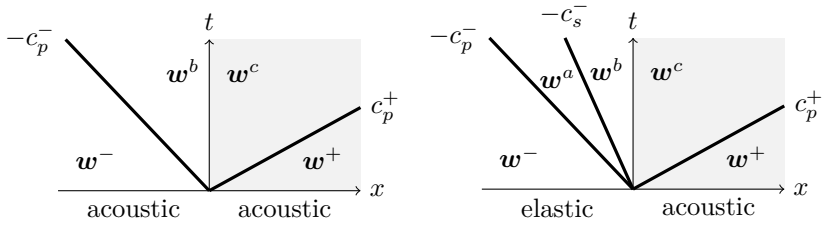


Figure 8: Solution structure of acoustic-acoustic and coupled acoustic-elastic Riemann problems.

Acoustic-acoustic interface

The jump conditions are read off from Figure 8.

$$\begin{aligned} \boldsymbol{w}^b - \boldsymbol{w}^- &= \alpha_1 \boldsymbol{r}_1^- \\ \boldsymbol{w}^+ - \boldsymbol{w}^c &= \alpha_9 \boldsymbol{r}_9^+ \end{aligned} \tag{4.38}$$

Having two unknowns we set the two boundary conditions: We require $\sigma_{xx}^b = \sigma_{xx}^c, u_1^b = u_1^c$, Hence, α_1 and α_9 are determined by

$$\begin{pmatrix} \lambda^- & \lambda^+ \\ c_p^- & -c_p^+ \end{pmatrix} \begin{pmatrix} \alpha_1 \\ \alpha_9 \end{pmatrix} = \begin{pmatrix} \sigma_{xx}^+ - \sigma_{xx}^- \\ u_1^+ - u_1^- \end{pmatrix}. \tag{4.39}$$

Elastic-acoustic interface

The jump conditions are read off from Figure 8.

$$\begin{aligned} \boldsymbol{w}^b - \boldsymbol{w}^- &= \alpha_1 \boldsymbol{r}_1^- + \alpha_2 \boldsymbol{r}_2^- + \alpha_3 \boldsymbol{r}_3^- \\ \boldsymbol{w}^+ - \boldsymbol{w}^c &= \alpha_9 \boldsymbol{r}_9^+ \end{aligned} \tag{4.40}$$

Having four unknowns we set the four boundary conditions. We require $\sigma_{xx}^b = \sigma_{xx}^c, \sigma_{xy}^b = \sigma_{xy}^c, \sigma_{xz}^b = \sigma_{xz}^c, u_1^b = u_1^c$. The unknowns are obtained from

$$\begin{pmatrix} \lambda^- + 2\mu^- & \lambda^+ & 0 & 0 \\ c_p^- & -c_p^+ & 0 & 0 \\ 0 & 0 & \mu^- & 0 \\ 0 & 0 & 0 & \mu^- \end{pmatrix} \begin{pmatrix} \alpha_1 \\ \alpha_9 \\ \alpha_2 \\ \alpha_3 \end{pmatrix} = \begin{pmatrix} \sigma_{xx}^+ - \sigma_{xx}^- \\ u_1^+ - u_1^- \\ \sigma_{xy}^+ - \sigma_{xy}^- \\ \sigma_{xz}^+ - \sigma_{xz}^- \end{pmatrix}. \tag{4.41}$$

The so-derived scheme has an interesting property. We have $\alpha_2 = -\sigma_{xy}^-/\mu^-$, as σ_{xy}^+ is initialised with zero in the acoustic domain, and is not changed in the course of a simulation. The interface state σ_{xy}^b and u_2^b are only affected by α_2, thus we get

$$\begin{pmatrix} \sigma_{xy}^b \\ u_2^b \end{pmatrix} = \begin{pmatrix} \sigma_{xy}^- \\ u_2^- \end{pmatrix} - \frac{\sigma_{xy}^-}{\mu^-} \begin{pmatrix} \mu^- \\ c_s^- \end{pmatrix} = \begin{pmatrix} 0 \\ -\frac{c_s^- \sigma_{xy}^-}{\mu^-} \end{pmatrix} \tag{4.42}$$

This shows that the shear stress $\sigma_{xy}^b = 0$ and that the tangential velocity component u_2^b is anti-parallel to σ_{xy}^b. A similar results holds for the pair σ_{xz}^b and u_3^b. Concluding, due to the discontinuous representation, non-zero shear traction at the boundary in the elastic domain may appear. However, due to the construction principle of the numerical flux, a surface velocity appears that penalises non-zero shear traction.

Acoustic-elastic interface

The analysis for the acoustic-elastic interface is quite similar to the elastic-acoustic case. We have

$$\begin{aligned} \boldsymbol{w}^b - \boldsymbol{w}^- &= \alpha_1 \boldsymbol{r}_1^- \\ \boldsymbol{w}^+ - \boldsymbol{w}^c &= \alpha_7 \boldsymbol{r}_7^+ + \alpha_8 \boldsymbol{r}_8^+ + \alpha_9 \boldsymbol{r}_9^+ \end{aligned} \tag{4.43}$$

We require $\sigma_{xx}^b = \sigma_{xx}^c, \sigma_{xy}^b = \sigma_{xy}^c, \sigma_{xz}^b = \sigma_{xz}^c, u_1^b = u_1^c$. The unknowns are obtained from

$$\begin{pmatrix} \lambda^- + 2\mu^- & \lambda^+ & 0 & 0 \\ c_p^- & -c_p^+ & 0 & 0 \\ 0 & 0 & \mu^+ & 0 \\ 0 & 0 & 0 & \mu^+ \end{pmatrix} \begin{pmatrix} \alpha_1 \\ \alpha_9 \\ \alpha_7 \\ \alpha_8 \end{pmatrix} = \begin{pmatrix} \sigma_{xx}^+ - \sigma_{xx}^- \\ u_1^+ - u_1^- \\ \sigma_{xy}^+ - \sigma_{xy}^- \\ \sigma_{xz}^+ - \sigma_{xz}^- \end{pmatrix}. \tag{4.44}$$

4.3 Boundary conditions

The jump conditions used in the construction of numerical fluxes may also be used to implement boundary conditions in the DG-framework. Essentially, given the values $\boldsymbol{w}^-$ at the boundary, one looks for artificial neighbouring data $\boldsymbol{w}^+$, such that the solution of the Riemann problem with initial data $\boldsymbol{w}^-$ and $\boldsymbol{w}^+$ yields a state $\boldsymbol{w}^b$ which satisfies the boundary conditions. Here we only treat boundary conditions for the elastic

wave equation. Boundary conditions for acoustics or the viscoelastic wave equation are handled analogously.

We first rescale the eigenvectors for convenience. The latter is not strictly necessary, but yields simpler formulas, and shows an analogy to the scheme of Duru et al. [42], too. We apply the scaling matrix

$$\operatorname{diag}\left(\frac{1}{c_p Z_p}, \frac{1}{c_s Z_s}, \frac{1}{c_s Z_s}, 1, 1, 1, \frac{1}{c_s Z_s}, \frac{1}{c_s Z_s}, \frac{1}{c_p Z_p}\right) \tag{4.45}$$

to R from the left, where we introduced the impedances $Z_i = \rho c_i$, and obtain

$$\mathfrak{R} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 - 2\frac{c_s Z_s}{c_p Z_p} & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 - 2\frac{c_s Z_s}{c_p Z_p} \\ 1 - 2\frac{c_s Z_s}{c_p Z_p} & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 - 2\frac{c_s Z_s}{c_p Z_p} \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ (Z_p)^{-1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & (-Z_p)^{-1} \\ 0 & (Z_s)^{-1} & 0 & 0 & 0 & 0 & 0 & (-Z_s)^{-1} & 0 \\ 0 & 0 & (Z_s)^{-1} & 0 & 0 & 0 & (-Z_s)^{-1} & 0 & 0 \end{pmatrix}. \tag{4.46}$$

We refer to the columns of $\mathfrak{R}$ with $\mathfrak{r}_1, \ldots, \mathfrak{r}_9$. The jump conditions are

$$\begin{aligned} \boldsymbol{w}^b - \boldsymbol{w}^- &= \alpha_1 \mathfrak{r}_1^- + \alpha_2 \mathfrak{r}_2^- + \alpha_3 \mathfrak{r}_3^-, \\ \boldsymbol{w}^+ - \boldsymbol{w}^c &= \alpha_7 \mathfrak{r}_7^+ + \alpha_8 \mathfrak{r}_8^+ + \alpha_9 \mathfrak{r}_9^+. \end{aligned} \tag{4.47}$$

From the jump condition we obtain the following equations:

$$\begin{aligned} \sigma_{xx}^b - \sigma_{xx}^- &= \alpha_1, & u_1^b - u_1^- &= \alpha_1 / Z_p^-, \\ \sigma_{xx}^+ - \sigma_{xx}^c &= \alpha_9, & u_1^+ - u_1^c &= -\alpha_9 / Z_p^+, \\ \sigma_{xy}^b - \sigma_{xy}^- &= \alpha_2, & u_2^b - u_2^- &= \alpha_2 / Z_s^-, \\ \sigma_{xy}^+ - \sigma_{xy}^c &= \alpha_8, & u_2^+ - u_2^c &= -\alpha_8 / Z_s^+, \\ \sigma_{xz}^b - \sigma_{xz}^- &= \alpha_3, & u_3^b - u_3^- &= \alpha_3 / Z_s^-, \\ \sigma_{xz}^+ - \sigma_{xz}^c &= \alpha_7, & u_3^+ - u_3^c &= -\alpha_7 / Z_s^+. \end{aligned} \tag{4.48}$$

The jump conditions show a structural similarity, as always two variables are coupled in the same way. Thus, we introduce the traction vector

$\boldsymbol{t} = (\sigma_{xx}, \sigma_{xy}, \sigma_{xz})$ and the impedance vector $\boldsymbol{Z} = (Z_p, Z_s, Z_s)$, which allows us to compactly write the jump conditions:

$$\forall i \in \{1,2,3\}: \quad \begin{aligned} t_i^b - t_i^- &= \alpha_i, & u_i^b - u_i^- &= \alpha_i / Z_i^-, \\ t_i^+ - t_i^c &= \alpha_{9-i+1}, & u_i^+ - u_i^c &= -\alpha_{9-i+1}/Z_i^+. \end{aligned} \tag{4.49}$$

4.3.1 Dynamic rupture

The presentation of dynamic rupture in this section is inspired by Duru et al. [42], but the results are mathematically equivalent to the original research for SeisSol [122–124].

Traction across a fault is required to be continuous. Thus we set $t_i^b = t_i^c =: \hat{t}_i$. The alphas are uniquely determined by the "hat"-traction. Writing the slip rate in terms of the "hat"-traction reveals the connection between slip rate and traction. To see this, let $[\![\hat{u}_i]\!] := u_i^c - u_i^b$ and $[\![u_i]\!] := u_i^+ - u_i^-$. The jump $[\![\hat{u}_i]\!]$ is computed to be

$$[\![\hat{u}_i]\!] = [\![u_i]\!] + \frac{\alpha_{9-i+1}}{Z_i^+} - \frac{\alpha_i}{Z_i^-} = [\![u_i]\!] + \frac{t_i^+ - \hat{t}_i}{Z_i^+} - \frac{\hat{t}_i - t_i^-}{Z_i^-} = [\![u_i]\!] + \frac{t_i^+}{Z_i^+} + \frac{t_i^-}{Z_i^-} - \frac{1}{\eta_i}\hat{t}_i, \tag{4.50}$$

where

$$\eta_i = \frac{Z_i^+ Z_i^-}{Z_i^+ + Z_i^-}. \tag{4.51}$$

Equation (4.50) allows us to compactly write

$$\hat{t}_i + \eta_i [\![\hat{u}_i]\!] = \theta_i, \tag{4.52}$$

with

$$\theta_i := \eta_i [\![u_i]\!] + \eta_i \left(\frac{t_i^+}{Z_i^+} + \frac{t_i^-}{Z_i^-} \right). \tag{4.53}$$

The quantities are already rotated to a fault-aligned system, cf. (4.5), such that the shear traction is given by $\hat{\boldsymbol{\tau}} = (\hat{t}_2, \hat{t}_3)$ and the slip rate by $\hat{\boldsymbol{s}} = ([\![\hat{u}_2]\!], [\![\hat{u}_3]\!])$. The normal stress is equal to the first component of traction, $\hat{\sigma}_n := \hat{t}_1$, and the slip rate in normal direction is set to zero, $[\![\hat{u}_1]\!] = 0$, as we ignore the possibility of fault opening. We thus obtain $\hat{t}_1 = \theta_1$. The relation between shear stress and slip rate (2.43) gives

$$\begin{aligned} \tau_S [\![\hat{u}_2]\!] - \hat{t}_2 \hat{s} &= 0, \\ \tau_S [\![\hat{u}_3]\!] - \hat{t}_3 \hat{s} &= 0, \end{aligned} \tag{4.54}$$

where $\hat{s} = \|\hat{\boldsymbol{s}}\| = \sqrt{[\![\hat{u}_2]\!]^2 + [\![\hat{u}_3]\!]^2}$. Inserting (4.54) into (4.52) gives:

$$\begin{aligned} [\![\hat{u}_2]\!] &= (\tau_S + \eta_2 \hat{s})^{-1} \hat{s} \theta_2, \\ [\![\hat{u}_3]\!] &= (\tau_S + \eta_3 \hat{s})^{-1} \hat{s} \theta_3, \end{aligned} \tag{4.55}$$

and hence (with $\eta_2 = \eta_3 =: \eta_s$ and $\tau_S, \eta_s > 0$)

$$\hat{s} = \hat{s}(\tau_S + \eta_s \hat{s})^{-1} \sqrt{\theta_2^2 + \theta_3^2}. \tag{4.56}$$

The latter equation has two solutions. Either $\hat{s} = 0$ or

$$\tau_S + \eta_s \hat{s} = \sqrt{\theta_2^2 + \theta_3^2}. \tag{4.57}$$

Which one of these solutions is correct is determined by $\hat{s} \geq 0$ and

$$\sqrt{\hat{t}_2^2 + \hat{t}_3^2} \leq \tau_S. \tag{4.58}$$

First, assume that $\hat{s} = 0$. Then, $\hat{t}_i = \theta_i$ and thus $\sqrt{\theta_2^2 + \theta_3^2} \leq \tau_S$ must be satisfied. Second, assume that $\hat{s} \neq 0$. Then, $0 < \eta_s \hat{s} = \sqrt{\theta_2^2 + \theta_3^2} - \tau_S$ and thus $\sqrt{\theta_2^2 + \theta_3^2} > \tau_S$ must be satisfied.

We remark that the fault strength may depend on $\hat{s}$, thus (4.57) is generally a non-linear equation. But for many friction laws, (4.57) has a unique solution [42]. For linear slip-weakening, the fault strength only depends on the state. As such, the slip rate my be stated explicitly.

$$\text{Linear slip-weakening law}: \quad \hat{s} = \max\left(0, \frac{\sqrt{\theta_2^2 + \theta_3^2} - \tau_S}{\eta_s}\right). \tag{4.59}$$

Having found the slip rate magnitude $\hat{s}$, we find the components of the slip rate vector with (4.55) and the "hat"-traction with (4.52). As the alphas are uniquely determined by the "hat"-traction, we obtain the following states on the fault:

$$\forall i \in \{1,2,3\}: \begin{array}{ll} t_i^b = \hat{t}_i, & u_i^b = u_i^- + (\hat{t}_i - t_i^-)/Z_i^-, \\ t_i^c = \hat{t}_i, & u_i^c = u_i^+ - (\hat{t}_i - t_i^+)/Z_i^+. \end{array} \tag{4.60}$$

4.3.2 Elastic-elastic interface

We have treated the regular elastic-elastic Riemann problem in Section 4.2.1. But it is interesting to note that the latter Riemann problem is a special

case of dynamic rupture when $\hat{s} = 0$. To see this, we solve the Riemann problem in the same manner as in Section 4.2.1, but with the scaled eigenvectors in $\mathfrak{R}$. The alphas can be shown to take the following values:

$$\forall i = \{1,2,3\}: \quad \alpha_i = \eta_i \left(\frac{[\![t_i]\!]}{Z_i^+} + [\![u_i]\!] \right) \quad \text{and} \quad \alpha_{9-i+1} = \eta_i \left(\frac{[\![t_i]\!]}{Z_i^-} - [\![u_i]\!] \right). \tag{4.61}$$

One computes

$$\forall i = \{1,2,3\}: \quad \theta_i - t_i^- = \alpha_i \quad \text{and} \quad t_i^+ - \theta_i = \alpha_{9-i+1}. \tag{4.62}$$

Hence, whenever $\hat{s} = 0$, then $\hat{t}_i = \theta_i$, and as such the dynamic rupture solution is identical to the solution of an elastic-elastic interface.

4.3.3 Free surface

A free surface boundary models the interface between solid and atmosphere. The influence of the atmosphere is neglected, such that the traction at the boundary is required to be zero. We set $\hat{t}_i = 0$ in (4.60) and obtain the state at the boundary:

$$\forall i \in \{1,2,3\}: \quad t_i^b = 0, \quad u_i^b = u_i^- - t_i^-/Z_i^-. \tag{4.63}$$

The alphas are here given by $\alpha_i = -t_i^-$ and $\alpha_{9-i+1} = t_i^+$, $i = 1,2,3$, whereas the alphas of a regular elastic-elastic interface are given by (4.61). By comparing terms we see that a Riemann problem with initial data $t_i^+ = -t_i^-$, $u_i^+ = u_i^-$, and $Z_i^+ = Z_i^-$ yields the exact same state at the boundary. Consequently, one may implement the free surface boundary condition by finding artificial neighbouring data $\boldsymbol{w}^+$ which yield the desired state at the boundary when plugged into the Riemann solver.

4.3.4 Absorbing boundaries

Lastly, we consider absorbing boundary conditions, which should allow waves to leave the finite simulation domain and should not reflect any waves. A simple implementation of absorbing boundary conditions is to ensure that the neighbouring state does not influence incoming waves [39]. As such, one sets $t_i^+ = 0$, $u_i^+ = 0$, and $Z_i^+ = Z_i^-$, and obtains the state

$$\forall i \in \{1,2,3\}: \quad t_i^b = \frac{1}{2} t_i^- - \frac{Z_i^-}{2} u_i^-, \quad u_i^b = \frac{1}{2} u_i^- - \frac{1}{2Z_i^-} \sigma_i^-. \tag{4.64}$$

CHAPTER

5

Semi-discrete stability

Numerical fluxes are presented in Chapter 4. Are these "good" fluxes and what justifies their use? The role of the numerical flux in a DG scheme is to ensure stability, which in combination with consistency guarantees convergence [68] (if a unique solution exists).

The stability of SeisSol's DG scheme has not been verified so far. In particular not for dynamic rupture simulations, in which a special numerical flux is used to impose the friction law on the fault [123, 124]. Hence, we investigate the stability of dynamic rupture simulations with an elastic rheological model.

A standard method to prove stability is the energy method: Wilcox et al. [162] derive a numerical flux for elastic-acoustic interfaces using Rankine-Hugeniot conditions, and prove stability using the energy method. Duru et al. [42] use the energy method to prove stability of a finite difference method for dynamic rupture simulations. In both papers, energy is defined as the sum of the kinetic energy and the strain energy [3]:

$$E(t) = \frac{1}{2} \int_{\Omega} \rho u_i u_i + \sigma_{ij} \epsilon_{ij} \, \mathrm{d}\boldsymbol{x}. \tag{5.1}$$

The authors then prove stability by showing that energy is bounded.

In this chapter, stability is proven with the energy method. The proof is quite technical, though. Thus, the main result is stated up front such that the busy reader may skip to the next chapter:

Theorem 1. *Assuming exact integration and in the absence of external body forces, the semi-discrete scheme for the elastic wave equation with heterogeneous material interfaces and dynamic rupture, free surface, and absorbing boundaries has non-increasing energy. That is,*

$$\frac{dE}{dt} \leq 0.$$

Therefore, the semi-discrete scheme is stable.

5.1 Rotational invariance revisited

In Section 4.1 it is claimed that the coefficient matrices ${A_{pq}}^d$ satisfy the rotational invariance property (4.4) for some matrix T, and the latter property is used to reduce plane-wave Riemann problems to a standard Riemann problem. We first review and prove the rotational invariance property for the elastic wave equation. Then, we prove the rotational invariance of the energy inner product, which is required in Section 5.2.

5.1.1 Rotation via isomorphism

The coefficient matrices ${A_{pq}}^d$, acting on the vector of quantities

$$\boldsymbol{q} = (\sigma_{xx}, \sigma_{yy}, \sigma_{zz}, \sigma_{xy}, \sigma_{yz}, \sigma_{xz}, u_1, u_2, u_3)^T, \tag{5.2}$$

are convenient for implementation but obscure the underlying concept of the strain and stress tensor. As a simplification, we make use of the isomorphism of $V^{3\times 3}_{\text{sym}} \oplus V^3$ and V^9, where V is a function space of sufficiently smooth functions [162].[1] The mapping between the two spaces is given by a bijective function φ, which maps every $(\boldsymbol{\sigma}, \boldsymbol{u}) \in V^{3\times 3}_{\text{sym}} \oplus V^3$ to a unique $\boldsymbol{q} \in V^9$, that is,

$$\begin{aligned} \boldsymbol{q} = \varphi(\boldsymbol{\sigma}, \boldsymbol{u}) &= (\sigma_{11}, \sigma_{22}, \sigma_{33}, \sigma_{12}, \sigma_{23}, \sigma_{13}, u_1, u_2, u_3)^T, \\ (\boldsymbol{\sigma}, \boldsymbol{u}) = \varphi^{-1}(\boldsymbol{q}) &= \left(\begin{pmatrix} q_1 & q_4 & q_6 \\ q_4 & q_2 & q_5 \\ q_6 & q_5 & q_3 \end{pmatrix}, \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} \right). \end{aligned} \tag{5.3}$$

The "rotation" of $\boldsymbol{q}$ is then defined via the rotation of $\boldsymbol{u}$ and the similarity transformation of $\boldsymbol{\sigma}$. That is, let $\mathfrak{T} := (\boldsymbol{n}, \boldsymbol{m}, \boldsymbol{l})$ be a rotation matrix,

[1] The $\oplus$-symbol denotes the direct sum of two spaces.

where the three vectors $\boldsymbol{n}, \boldsymbol{m}, \boldsymbol{l}$ span an orthonormal basis in three dimensions. Then, the vector $\boldsymbol{u}$ is rotated with $\mathfrak{T}\boldsymbol{u}$ and the tensor σ is transformed via $\mathfrak{T}\boldsymbol{\sigma}\mathfrak{T}^T$. We define

$$T\boldsymbol{q} = \varphi\left(\mathfrak{T}\left(\varphi^{-1}(\boldsymbol{q})\right)\right), \quad \text{with} \quad \mathfrak{T}(\boldsymbol{\sigma}, \boldsymbol{u}) = (\mathfrak{T}\boldsymbol{\sigma}\mathfrak{T}^T, \mathfrak{T}\boldsymbol{u}). \tag{5.4}$$

One easily checks that φ, φ^{-1}, and $\mathfrak{T}(\boldsymbol{\sigma}, \boldsymbol{u})$ are linear maps, hence $T\boldsymbol{q}$ is a linear map. Consequently, we may find a matrix T which represents the linear map $T\boldsymbol{q}$. We introduce the following shorthand notation for (5.4):

$$T\boldsymbol{q} \quad \simeq \quad (\mathfrak{T}\boldsymbol{\sigma}\mathfrak{T}^T, \mathfrak{T}\boldsymbol{u}) \tag{5.5}$$

The inverse of T is found in a similar way, using that $\mathfrak{T}^{-1} = \mathfrak{T}^T$:

$$T^{-1}\boldsymbol{q} \quad \simeq \quad (\mathfrak{T}^T\boldsymbol{\sigma}\mathfrak{T}, \mathfrak{T}^T\boldsymbol{u}) \tag{5.6}$$

We note that $T^{-1} \neq T^T$. Hence, rotations are easy to handle in $V^{3\times3}_{\text{sym}} \oplus V^3$, but not in V^9.

5.1.2 Rotational invariance of the plane wave operator

The coefficient matrices ${A_{pq}}^d$ have to be constructed such that the elastic wave equation, (2.8), and the general system of linear hyperbolic PDEs, (3.1), are equivalent. As ${A_{pq}}^d$ collects the coefficient that acts on the d-th derivative, we formally replace $\frac{\partial}{\partial x_j}$ with δ_{dj} (and $\frac{\partial}{\partial t}$ with zero) in (2.8), which yields

$$\begin{aligned} (\mathcal{A}^{1d}\boldsymbol{u})_{ij} &:= -\lambda\delta_{ij}\delta_{dk}u_k - \mu\left(\delta_{dj}u_i + \delta_{di}u_j\right), \\ (\mathcal{A}^{2d}\boldsymbol{\sigma})_i &:= -\delta_{dj}\sigma_{ji}. \end{aligned} \tag{5.7}$$

and as such obtain the definition of ${A_{pq}}^d$ via

$${A_{::}}^d\boldsymbol{q} \quad \simeq \quad (\mathcal{A}^{1d}\boldsymbol{u}, \mathcal{A}^{2d}\boldsymbol{\sigma}). \tag{5.8}$$

The plane wave operator $n_d{A_{pq}}^d$ is isomorphic,

$$n_d{A_{::}}^d\boldsymbol{q} \quad \simeq \quad (n_d\mathcal{A}^{1d}\boldsymbol{u}, n_d\mathcal{A}^{2d}\boldsymbol{\sigma}), \tag{5.9}$$

as well as

$$T{A_{::}}^1T^{-1}\boldsymbol{q} \quad \simeq \quad \left(\mathfrak{T}\left(\mathcal{A}^{11}\left(\mathfrak{T}^T\boldsymbol{u}\right)\right)\mathfrak{T}^T, \mathfrak{T}\left(\mathcal{A}^{21}\left(\mathfrak{T}^T\boldsymbol{\sigma}\mathfrak{T}\right)\right)\right). \tag{5.10}$$

So showing the equality of the right-hand sides of (5.9) and (5.10) proves the rotational invariance property (4.4):

Lemma 1. *The plane wave operator is rotational invariant, that is,*

$$(n_d\mathcal{A}^{1d}\boldsymbol{u}, n_d\mathcal{A}^{2d}\boldsymbol{\sigma}) = \left(\mathcal{T}\left(\mathcal{A}^{11}\left(\mathcal{T}^T\boldsymbol{u}\right)\right)\mathcal{T}^T, \mathcal{T}\left(\mathcal{A}^{21}\left(\mathcal{T}^T\boldsymbol{\sigma}\mathcal{T}\right)\right)\right).$$

Proof. The left-hand particle velocity vector is computed to be equal to the right-hand particle velocity vector:

$$\begin{aligned}(n_d\mathcal{A}^{2d}\boldsymbol{\sigma})_i &= -n_d\delta_{dj}\sigma_{ji} = -n_j\sigma_{ji},\\ \left(\mathcal{T}\left(\mathcal{A}^{21}\left(\mathcal{T}^T\boldsymbol{\sigma}\mathcal{T}\right)\right)\right)_i &= -\mathcal{T}_{ik}\delta_{1j}\mathcal{T}_{pj}\sigma_{pq}\mathcal{T}_{qk} = -\delta_{iq}\mathcal{T}_{p1}\sigma_{pq} = -n_p\sigma_{pi},\end{aligned}$$

where we used $\mathcal{T}_{p1} = n_p$ and $\mathcal{T}_{ik}\mathcal{T}_{qk} = \delta_{iq}$.

The left-hand stress tensor is

$$\begin{aligned}(n_d\mathcal{A}^{1d}\boldsymbol{u})_{ij} &= -n_d\lambda\delta_{ij}\delta_{dk}u_k - \mu n_d(\delta_{dj}u_i + \delta_{di}u_j)\\ &= -\lambda\delta_{ij}n_ku_k - \mu(n_ju_i + n_iu_j).\end{aligned}$$

And the right-hand stress tensor is computed to be equal to the left-hand stress tensor:

$$\begin{aligned}&\left(\mathcal{T}\left(\mathcal{A}^{11}\left(\mathcal{T}^T\boldsymbol{u}\right)\right)\mathcal{T}^T\right)_{ij}\\ &\quad= \mathcal{T}_{ip}\left(-\lambda\delta_{pq}\delta_{1k}\mathcal{T}_{rk}u_r - \mu(\delta_{1q}\mathcal{T}_{rp}u_r + \delta_{1p}\mathcal{T}_{rq}u_r)\right)\mathcal{T}_{jq}\\ &\quad= -\lambda\delta_{ij}\delta_{1k}\mathcal{T}_{rk}u_r - \mu(\mathcal{T}_{j1}\delta_{ir}u_r + \mathcal{T}_{i1}\delta_{jr}u_r)\\ &\quad= -\lambda\delta_{ij}n_ru_r - \mu(n_ju_i + n_iu_j).\end{aligned}$$

□

5.1.3 The energy inner product

The constitutive tensor c_{ijkl}, defined in (2.6), can be inverted. That is, we find a tensor s_{ijkl}, such that $s_{ijkl}c_{klpq} = \frac{1}{2}(\delta_{ip}\delta_{jq} + \delta_{iq}\delta_{jp})$. With tensor s_{ijkl} we may express strain in terms of stress,

$$\epsilon_{ij} = s_{ijkl}\sigma_{kl} = -\frac{\lambda}{2\mu(3\lambda+2\mu)}\delta_{ij}\sigma_{kk} + \frac{1}{2\mu}\sigma_{ij}. \tag{5.11}$$

The energy integral (5.1) becomes

$$E(t) = \frac{1}{2}\int_\Omega \rho u_i u_i + \sigma_{ij} s_{ijkl} \sigma_{kl} \,\mathrm{d}\boldsymbol{x}$$
$$= \frac{1}{2}\int_\Omega \rho u_i u_i - \frac{\lambda}{2\mu(3\lambda+2\mu)}(\sigma_{kk})^2 + \frac{1}{2\mu}\sigma_{ij}\sigma_{ij}\,\mathrm{d}\boldsymbol{x}. \quad (5.12)$$

Equation (5.12) is positive definite, and thus $2E(t)$ is the norm induced by the following inner product on $V^{3\times 3}_{\mathrm{sym}} \oplus V^3$:

$$\langle(\boldsymbol{\tau},\boldsymbol{v}),(\boldsymbol{\sigma},\boldsymbol{u})\rangle = \int_\Omega \tau_{ij} s_{ijkl}\sigma_{kl} + \rho v_i u_i \,\mathrm{d}\boldsymbol{x}. \quad (5.13)$$

Lemma 2. *The inner product defined by* (5.13) *is rotational invariant, that is,*

$$\langle(\boldsymbol{\tau},\boldsymbol{v}),(\boldsymbol{\sigma},\boldsymbol{u})\rangle = \langle(\boldsymbol{\tau}',\boldsymbol{v}'),(\boldsymbol{\sigma}',\boldsymbol{u}')\rangle\,,$$

where $(\boldsymbol{\tau}',\boldsymbol{v}') = (\mathcal{T}^T\boldsymbol{\tau}\mathcal{T}, \mathcal{T}^T\boldsymbol{v})$ *and* $(\boldsymbol{\sigma}',\boldsymbol{u}') = (\mathcal{T}^T\boldsymbol{\sigma}\mathcal{T}, \mathcal{T}^T\boldsymbol{u})$.

Proof. To simplify notation, we write $s_{ijkl} = a\delta_{ij}\delta_{kl} + b(\delta_{ik}\delta_{jl} + \delta_{il}\delta_{jk})$, with $a = -\frac{\lambda}{2\mu(3\lambda+2\mu)}$ and $b = \frac{1}{4\mu}$. We check

$$\langle(\boldsymbol{\tau}',\boldsymbol{v}'),(\boldsymbol{\sigma}',\boldsymbol{u}')\rangle = \int_\Omega \mathcal{T}_{pi}\tau_{pq}\mathcal{T}_{qj} s_{ijkl}\mathcal{T}_{rk}\sigma_{rs}\mathcal{T}_{sl} + \rho\mathcal{T}_{pi}v_p\mathcal{T}_{qi}u_q\,\mathrm{d}\boldsymbol{x}$$
$$= \int_\Omega \mathcal{T}_{pi}\tau_{pq}\mathcal{T}_{qj}\left(a\delta_{ij}\delta_{kl} + b(\delta_{ik}\delta_{jl}+\delta_{il}\delta_{jk})\right)\mathcal{T}_{rk}\sigma_{rs}\mathcal{T}_{sl} + \rho\mathcal{T}_{pi}v_p\mathcal{T}_{qi}u_q\,\mathrm{d}\boldsymbol{x}$$
$$= \int_\Omega \tau_{pq}\left(a\delta_{pq}\delta_{kl} + b(\mathcal{T}_{pk}\mathcal{T}_{ql}+\mathcal{T}_{pl}\mathcal{T}_{qk})\right)\mathcal{T}_{rk}\sigma_{rs}\mathcal{T}_{sl} + \rho v_p\delta_{pq}u_q\,\mathrm{d}\boldsymbol{x}$$
$$= \int_\Omega \tau_{pq}\left(a\delta_{pq}\delta_{rs} + b(\delta_{pr}\delta_{qs}+\delta_{ps}\delta_{qr})\right)\sigma_{rs} + \rho v_p u_p\,\mathrm{d}\boldsymbol{x}$$
$$= \int_\Omega \tau_{pq}s_{pqrs}\sigma_{rs} + \rho v_p u_p = \langle(\boldsymbol{\tau},\boldsymbol{v}),(\boldsymbol{\sigma},\boldsymbol{u})\rangle$$

□

We find the inner product for $\boldsymbol{q} \in V^9$ corresponding to (5.13) using the isomorphism φ:

$$\langle \boldsymbol{p},\boldsymbol{q}\rangle = \left\langle \varphi^{-1}(\boldsymbol{p}), \varphi^{-1}(\boldsymbol{q})\right\rangle. \quad (5.14)$$

One can show by a short computation that

$$\langle \boldsymbol{p}, \boldsymbol{q} \rangle = \int_\Omega \boldsymbol{p} \cdot (\mathcal{P}\boldsymbol{q}) \, \mathrm{d}\boldsymbol{x}, \tag{5.15}$$

where

$$\mathcal{P} := \begin{pmatrix} \frac{\lambda+\mu}{\mu(3\lambda+2\mu)} & -\frac{\lambda}{2\mu(3\lambda+2\mu)} & -\frac{\lambda}{2\mu(3\lambda+2\mu)} & 0 & 0 & 0 & 0 & 0 & 0 \\ -\frac{\lambda}{2\mu(3\lambda+2\mu)} & \frac{\lambda+\mu}{\mu(3\lambda+2\mu)} & -\frac{\lambda}{2\mu(3\lambda+2\mu)} & 0 & 0 & 0 & 0 & 0 & 0 \\ -\frac{\lambda}{2\mu(3\lambda+2\mu)} & -\frac{\lambda}{2\mu(3\lambda+2\mu)} & \frac{\lambda+\mu}{\mu(3\lambda+2\mu)} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{\mu} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{\mu} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{\mu} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \rho & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \rho & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \rho \end{pmatrix}. \tag{5.16}$$

Note that energy is given by $E(t) = \frac{1}{2}\langle \boldsymbol{q}, \boldsymbol{q} \rangle$, and that the rotational invariance property becomes

$$\left\langle T^{-1}\boldsymbol{p}, T^{-1}\boldsymbol{q} \right\rangle = \int_\Omega (T^{-1}\boldsymbol{p}) \cdot (\mathcal{P}T^{-1}\boldsymbol{q}) \, \mathrm{d}\boldsymbol{x} = \langle \boldsymbol{p}', \boldsymbol{q}' \rangle . \tag{5.17}$$

The matrix $\mathcal{P}$ is symmetric, diagonal dominant and thus positive definite. In addition, the products of $\mathcal{P}$ with the coefficient matrices ${A_{pq}}^d$ are symmetric and independent of the material parameters. In particular for $d = 1$ we have

$$\mathcal{P}{A_{::}}^1 = -\begin{pmatrix} 0&0&0&0&0&0&1&0&0 \\ 0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&1&0 \\ 0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&1 \\ 1&0&0&0&0&0&0&0&0 \\ 0&0&0&1&0&0&0&0&0 \\ 0&0&0&0&0&1&0&0&0 \end{pmatrix}. \tag{5.18}$$

The cases $d = 2$ and $d = 3$ are given by

$$\mathcal{P}{A_{::}}^2 = -\begin{pmatrix} 0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&1&0 \\ 0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&1&0&0 \\ 0&0&0&0&0&0&0&0&0&1 \\ 0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&1&0&0&0&0&0 \\ 0&1&0&0&0&0&0&0&0 \\ 0&0&0&0&1&0&0&0&0 \end{pmatrix}, \quad \mathcal{P}{A_{::}}^3 = -\begin{pmatrix} 0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&1 \\ 0&0&0&0&0&0&0&0&1&0 \\ 0&0&0&0&0&0&0&1&0&0 \\ 0&0&0&0&0&0&1&0&0&0 \\ 0&0&0&0&0&1&0&0&0&0 \\ 0&0&1&0&0&0&0&0&0&0 \end{pmatrix}. \tag{5.19}$$

5.2 Energy estimate

Energy is directly derived from the weak form. We prefer to work with the weak form from (3.3) instead of the processed weak form (3.25). Both forms are mathematically equivalent for linear numerical fluxes, and also equivalent for non-linear fluxes if we assume exact integration. That is, we begin with

$$\int_{\mathcal{T}_m} \phi_k \frac{\partial q_p}{\partial t} \,\mathrm{d}\boldsymbol{x} + \int_{\partial\mathcal{T}_m} \phi_k \left(n_d A_{pq}{}^d q_q\right)^* \,\mathrm{d}S - \int_{\mathcal{T}_m} \frac{\phi_k}{\partial x_d} A_{pq}{}^d q_q \,\mathrm{d}\boldsymbol{x} = 0. \tag{5.20}$$

Recall that the quantities may be expanded in terms of basis functions, i.e. $q_p = Q_{lp}\phi_l$. Multiplying the weak form with $Q_{kr}\mathcal{P}_{rp}$ and summing over r, k, and p yields

$$\int_{\mathcal{T}_m} q_r \mathcal{P}_{rp} \frac{\partial q_p}{\partial t} \,\mathrm{d}\boldsymbol{x} + \int_{\partial\mathcal{T}_m} q_r \mathcal{P}_{rp} \left(n_d A_{pq}{}^d q_q\right)^* \,\mathrm{d}S - \int_{\mathcal{T}_m} \frac{q_r}{\partial x_d} \mathcal{P}_{rp} A_{pq}{}^d q_q \,\mathrm{d}\boldsymbol{x} = 0. \tag{5.21}$$

We integrate the right-most term by parts and obtain the strong form

$$\begin{aligned} &\int_{\mathcal{T}_m} q_r \mathcal{P}_{rp} \frac{\partial q_p}{\partial t} \,\mathrm{d}\boldsymbol{x} + \int_{\partial\mathcal{T}_m} q_r \mathcal{P}_{rp} \left(n_d A_{pq}{}^d q_q\right)^* \,\mathrm{d}S \\ &\quad - \int_{\partial\mathcal{T}_m} q_r \mathcal{P}_{rp} n_d A_{pq}{}^d q_q \,\mathrm{d}S + \int_{\mathcal{T}_m} q_r \mathcal{P}_{rp} A_{pq}{}^d \frac{\partial q_q}{\partial x_d} \,\mathrm{d}\boldsymbol{x} = 0. \end{aligned} \tag{5.22}$$

Adding the weak form and the strong form yields

$$2\int_{\mathcal{T}_m} q_r \mathcal{P}_{rp} \frac{\partial q_p}{\partial t} \,\mathrm{d}\boldsymbol{x} + \int_{\partial\mathcal{T}_m} q_r \mathcal{P}_{rp} \left(2\left(n_d A_{pq}{}^d q_q\right)^* - n_d A_{pq}{}^d q_q\right) \mathrm{d}S = 0. \tag{5.23}$$

The stiffness term vanishes, due to the symmetry $\mathcal{P}_{rp}A_{pq}{}^d = \mathcal{P}_{qp}A_{pr}{}^d$.

The left-most integral is identical with 2 times the element-local energy rate, which follows from the symmetry of $\mathcal{P}$ and the product rule ($\frac{\partial}{\partial t}(q_r\mathcal{P}_{rp}q_p) = 2q_r\mathcal{P}_{rp}\frac{\partial q_p}{\partial t}$). The total energy rate is obtained by summing over all elements:

$$\frac{\partial E}{\partial t} = \sum_m - \int_{\partial\mathcal{T}_m} q_r \mathcal{P}_{rp} \left(\left(n_d A_{pq}{}^d q_q\right)^* - \frac{1}{2} n_d A_{pq}{}^d q_q\right) \mathrm{d}S. \tag{5.24}$$

The right-hand side now only depends on the numerical flux. As every face is shared by at most two elements, we only need to analyse the in- and outflow at an interface, and we may analyse every interface separately, i.e.

$$\frac{\partial E}{\partial t} = - \sum_{(m,n)\in \text{Fault}} E_{mn}^{\text{dynamic rupture}} - \sum_{(m,n)\in \text{Regular}} E_{mn}^{\text{elastic}} - \sum_{(m,n)\in \text{Absorbing}} E_{mn}^{\text{absorbing}} - \sum_{(m,n)\in \text{Free surface}} E_{mn}^{\text{free surface}}. \quad (5.25)$$

5.3 Stability of the numerical flux

We first analyse the in- and outflow at dynamic rupture interfaces. From Chapter 4 we know that the numerical flux is given by

$$(n_d A_{pq}{}^d q_q)^* = n_d A_{pq}{}^d q_q^b = T_{pr} A_{rs}{}^1 T_{sq}^{-1} q_q^b \quad (5.26)$$

where we used Lemma 1 in the last equality. The energy contribution of a dynamic rupture interface is given by the sum of the flux on both sides. We w.l.o.g. denote element m the "$-$"-element and element n the "$+$"-element, and let normal point from the $-$ side to the $+$ side. The energy rate contribution from the numerical flux is

$$E_{mn}^{\text{dynamic rupture}} = \int_{\partial\mathcal{T}_m \cap \partial\mathcal{T}_n} q_r^- {}^m\mathcal{P}_{rp} T_{pr} {}^m A_{rs}{}^1 T_{sq}^{-1} \left(q_q^b - \frac{1}{2} q_q^- \right) - q_r^+ {}^n\mathcal{P}_{rp} T_{pr} {}^n A_{rs}{}^1 T_{sq}^{-1} \left(q_q^c - \frac{1}{2} q_q^+ \right) \mathrm{d}S. \quad (5.27)$$

The numerical flux is formulated in terms of face-aligned coordinates, that is, $\boldsymbol{w}^\pm = T^{-1}\boldsymbol{q}^\pm$. Using the rotational invariance of the energy inner product from Lemma 2 (consider also (5.17)), we obtain

$$E_{mn}^{\text{dynamic rupture}} = \int_{\partial\mathcal{T}_m \cap \partial\mathcal{T}_n} w_r^- {}^m\mathcal{P}_{rp} {}^m A_{rs}{}^1 \left(w_s^b - \frac{1}{2} w_s^- \right) - w_r^+ {}^n\mathcal{P}_{rp} {}^n A_{rs}{}^1 \left(w_s^c - \frac{1}{2} w_s^+ \right) \mathrm{d}S. \quad (5.28)$$

In the following, we label the components of $\boldsymbol{w}$-vectors with

$$\boldsymbol{w} = (t_1, \blacksquare, \blacksquare, t_2, \blacksquare, t_3, u_1, u_2, u_3)^T. \quad (5.29)$$

Some components are marked with ✖. These components lie in the null space of $\mathcal{P}A_{::}{}^{1}$ and we do not care about them. (Note that we identify $(t_1, t_2, t_3) = (\sigma_{xx}, \sigma_{xy}, \sigma_{xz})$, as in Section 4.3.) Using (5.18) we explicitly state the energy contribution as

$$\int_{\partial\mathcal{T}_m \cap \partial\mathcal{T}_n} \sum_{i=1}^{3} -t_i^- \left(u_i^b - \frac{1}{2}u_i^-\right) - u_i^- \left(t_i^b - \frac{1}{2}t_i^-\right) + t_i^+ \left(u_i^c - \frac{1}{2}u_i^+\right) + u_i^+ \left(t_i^c - \frac{1}{2}t_i^+\right) \mathrm{d}S. \quad (5.30)$$

We omit the integral and the sum sign for brevity. Inserting the boundary states from (4.60) yields

$$\begin{aligned} &- t_i^- \left(u_i^- + \frac{\hat{t}_i - t_i^-}{Z_i^-} - \frac{1}{2}u_i^-\right) - u_i^- \left(\hat{t}_i - \frac{1}{2}t_i^-\right) \\ &\quad + t_i^+ \left(u_i^+ - \frac{\hat{t}_i - t_i^+}{Z_i^+} - \frac{1}{2}u_i^+\right) + u_i^+ \left(\hat{t}_i - \frac{1}{2}t_i^+\right) \\ &\quad = [\![u_i]\!]\hat{t}_i - t_i^- \frac{\hat{t}_i - t_i^-}{Z_i^-} - t_i^+ \frac{\hat{t}_i - t_i^+}{Z_i^+} \end{aligned} \quad (5.31)$$

With the identities

$$\begin{aligned} \hat{t}_i - t_i^- &= \theta_i - \eta_i [\![\hat{u}_i]\!] - t_i^- = \eta_i \frac{[\![t_i]\!]}{Z_i^+} + \eta_i \left([\![u_i]\!] - [\![\hat{u}_i]\!]\right), \\ \hat{t}_i - t_i^+ &= \theta_i - \eta_i [\![\hat{u}_i]\!] - t_i^+ = -\eta_i \frac{[\![t_i]\!]}{Z_i^-} + \eta_i \left([\![u_i]\!] - [\![\hat{u}_i]\!]\right), \end{aligned} \quad (5.32)$$

and the identity

$$\frac{t_i^+}{Z_i^+} + \frac{t_i^-}{Z_i^-} = \frac{\hat{t}_i}{\eta_i} + [\![\hat{u}_i]\!] - [\![u_i]\!], \quad (5.33)$$

Equation (5.31) becomes

$$\begin{aligned} &[\![u_i]\!]\hat{t}_i - \frac{t_i^-}{Z_i^-}\eta_i \left(\frac{[\![t_i]\!]}{Z_i^+} + [\![u_i]\!] - [\![\hat{u}_i]\!]\right) - \frac{t_i^+}{Z_i^+}\eta_i \left(-\frac{[\![t_i]\!]}{Z_i^-} + [\![u_i]\!] - [\![\hat{u}_i]\!]\right) \\ &\quad = [\![\hat{u}_i]\!]\hat{t}_i + \eta_i \left([\![u_i]\!] - [\![\hat{u}_i]\!]\right)^2 + \frac{\eta_i}{Z_i^- Z_i^+}[\![t_i]\!]^2. \end{aligned} \quad (5.34)$$

It follows that

$$E_{mn}^{\text{dynamic rupture}} = \int \sum_{i=1}^{3} [\![\hat{u}_i]\!]\hat{t}_i + \eta_i \left([\![u_i]\!] - [\![\hat{u}_i]\!]\right)^2 + \frac{\eta_i}{Z_i^- Z_i^+} [\![t_i]\!]^2 \,\mathrm{d}S. \quad (5.35)$$

Moreover, the interface condition (2.43) is ensured in Section 4.3.1. Thus,

$$E_{mn}^{\text{dynamic rupture}} = \int \tau_S \hat{s} + \sum_{i=1}^{3} \eta_i \left([\![u_i]\!] - [\![\hat{u}_i]\!]\right)^2 + \frac{\eta_i}{Z_i^- Z_i^+} [\![t_i]\!]^2 \,\mathrm{d}S. \quad (5.36)$$

It is shown in Section 4.3.2 that the elastic-elastic interfaces are a special case of dynamic rupture interfaces with slip-rate zero. Therefore,

$$E_{mn}^{\text{elastic}} = \int \sum_{i=1} \eta_i [\![u_i]\!]^2 + \frac{\eta_i}{Z_i^- Z_i^+} [\![t_i]\!]^2 \,\mathrm{d}S. \quad (5.37)$$

For free surface boundaries one may show that

$$E_{mn}^{\text{free surface}} = \int \sum_{i=1} \frac{1}{Z_i^-} (t_i^-)^2 \,\mathrm{d}S. \quad (5.38)$$

and for absorbing boundaries one may show that

$$E_{mn}^{\text{absorbing}} = \int \sum_{i=1} \frac{1}{2Z_i^-} (t_i^-)^2 + \frac{Z_i^-}{2} (u_i^-)^2 \,\mathrm{d}S. \quad (5.39)$$

Proof of Theorem 1. $E_{mn}^{\text{absorbing}}, E_{mn}^{\text{absorbing}}, E_{mn}^{\text{elastic}}$, and $E_{mn}^{\text{dynamic rupture}}$ are non-negative, because $\tau_S \geq 0$ and $\hat{s} \geq 0$. Stability follows from (5.25). □

5.4 A brief note on pre-stress

It was shown in the last section, that energy is non-increasing. Hence, the initial energy must be non-zero, because otherwise one would compute only zeros. Energy is introduced into the simulation by prescribing a pre-stress, and we briefly discuss two ways of doing this.

One method is to project pre-stress on the degrees of freedom at $t = 0$. However, one needs to be careful not to generate spurious waves at the absorbing or free surface boundaries. Moreover, the stress field needs to be known throughout the whole domain.

Another method is to solve for stress perturbation only. That is, one separates stress in $\sigma = \overset{\circ}{\sigma} + \Delta\sigma$. Pre-stress $\overset{\circ}{\sigma}$ is continuous everywhere and is in static equilibrium. The semi-discrete scheme of the perturbed quantities $\Delta \boldsymbol{q}$ is obtained by subtracting the semi-discrete scheme of $\overset{\circ}{\boldsymbol{q}}$ from the semi-discrete of $\boldsymbol{q} = \overset{\circ}{\boldsymbol{q}} + \Delta \boldsymbol{q}$. Stability then follows immediately because both semi-discrete schemes are stable due to Theorem 1. Due to linearity, most of the terms cancel and $\overset{\circ}{\boldsymbol{q}}$ does not appear anymore. The friction law is the only exception. Here, normal pre-stress is added in the calculation of the fault strength, $\tau_S = \max(0, -(\hat{\sigma}_n + \overset{\circ}{\sigma}_n) f(\|\boldsymbol{s}\|, \psi))$, and the shear pre-stresses are added to θ_2 and θ_3 in the computation of the slip-rate.

5.5 Discussion

The energy estimate illustrates the role of the numerical flux. For elastic-elastic interfaces, penalties are added if particle velocity or traction are discontinuous at an interface, i.e. if they violate the physical interface condition. Similarly, non-zero traction is penalised for free surface interfaces. Absorbing boundaries allow energy to leave the domain. On the fault, the energy rate mimics the physical energy rate [42] plus additional terms, which penalise deviations from the imposed slip rate and penalise discontinuous traction.

CHAPTER

6

Yet another tensor toolbox

This chapter is an extended version of the following manuscript, which is currently under review for publication:

Carsten Uphoff et al. "Yet Another Tensor Toolbox for discontinuous Galerkin methods and other applications". arXiv:1903.11521. Submitted to ACM Transactions on Mathematical Software.

In an earthquake simulation using the ADER-DG method presented in Chapter 3, almost all of the computation time is spent in computing small tensor[1] contractions. Optimising these is the key ingredient to achieve a high single node performance.

In previous work the ADER-DG method for the elastic wave equation is optimised, where only small matrix matrix multiplications are required. These can in principle be implemented using the General Matrix-Matrix Multiplication (GEMM) operation, which is available in any implementation of the Basic Linear Algebra Subprograms (BLAS) specification. The GEMM operation is typically optimised for large matrix matrix multiplications, such that many implementations are sub-optimal for small matrix matrix multiplications [66], which lead to the development of specialised code generators for SeisSol [19, 22, 64, 67]. The code generators have

[1] We note that the term "tensor" is used to refer to multi-dimensional arrays throughout this work. There is no distinction between lower (covariant) and upper (contravariant) indices, which is common practice in differential geometry [35].

been incorporated in the library LIBXSMM [66], which is able to generate GEMMs of any size and is optimised for small matrix multiplications for all recent Intel architectures.

In this chapter, a domain-specific language (DSL) is developed for small tensor contractions, with the ultimate goal to reduce the development time for the implementation of advanced earthquake models without sacrificing the excellent performance achieved in previous work. The scope of the DSL is deliberately more general than small matrix matrix multiplications, in order to support a large class of applications. For example, the tensor structure of viscoelastic attenuation [156] may be modelled as small tensor contractions with three-dimensional tensors (see Section 7.3). Another example are ensemble simulations as discussed by Breuer et al. [20], where the degrees of freedom effectively become a three-dimensional tensor (or a four-dimensional tensor for viscoelastic attenuation). Small tensor contractions are also used in many other applications. For example in computational fluid dynamics [69, 159], spectral element methods [69], quantum chemistry [146], or in the assembly of finite elements [84, 106].

A large body of literature exists on the efficient implementation of tensor contractions, where one has to distinguish two classes of papers: The first class considers the optimal implementation of binary tensor contractions [101, 109, 139, 142, 144, 146], which can be further subdivided in nested loops, Loop-over-GEMM, and Transpose-Transpose-GEMM-Transpose [144]. The second class of papers discusses the optimisation of expressions involving several tensors. E.g. an optimisation step is typically the minimisation of floating point operations [97]. The Tensor Contraction Engine [9] is a prominent representative of the second class of papers. The latter has been used in the computational chemistry community, but targets very large tensors which even requires storing intermediate results on hard disks [9]. For GPUs, there exists the Barracuda framework [114]. Optimisation techniques for tensor contractions are also found in the COFFEE compiler [105, 106].

So, why do we need **Y**et **A**nother **Te**nsor **To**olbox (YATeTo)? The reason lies within the set of assumptions we make about the target applications, which are motivated by the ADER-DG method for linear hyperbolic problems:

— Tensors are small enough to fit into memory or last-level caches. There is no need to account for memory constraints [9].

— Several tensors are sparse. Sparsity patterns and tensor shapes are known at compile time. It is feasible to deal with sparsity explicitly instead of having to estimate sparsity [96].

— Intermediate reorganisation of data is not beneficial and should be avoided. (That is, in contrast to large GEMMs [57], one cannot amortise copying data to temporary arrays.)

— Software prefetching is necessary on some architectures for best performance [64, 158].

— Highly efficient small GEMMs (for dense and sparse matrices) are available as library or can be generated generically.

The last assumption is fundamental to this work, as eventually most of the operation shall be mapped to small GEMMs. The latter also distinguishes this work from loop-transformation approaches [84, 105, 106], which eventually generate C or C++ code and rely on a general purpose compiler to emit machine code. YATeTo is agnostic about the GEMM back-end, as it may include inline assembly generated by external tools or calls to external libraries.

YATeTo is written in Python 3 and is licensed under the 3-Clause BSD License. The code is available on `www.github.com/SeisSol/yateto`.

6.1 Language definition

The DSL is based on Einstein's convention, which originated from a short paragraph in an early work on general relativity:

> "Dafür führen wir die Vorschrift ein: Tritt ein Index in einem Term eines Ausdrucks zweimal auf, so ist über ihn stets zu summieren, wenn nicht ausdrücklich das Gegenteil bemerkt ist." [46]

Loosely translated, the convention implies a summation whenever an index appears twice, if not noted otherwise. For instance, matrix multiplication which is formally written in the following way

$$\forall (i,j) \in [1\,..\,I] \times [1\,..\,J] : C_{ij} = \sum_{k=1}^{K} A_{ik} B_{kj}, \tag{6.1}$$

where $C \in \mathbb{R}^{I \times J}$, $A \in \mathbb{R}^{I \times K}$, and $B \in \mathbb{R}^{K \times J}$, is abbreviated with

$$C_{ij} = A_{ik} B_{kj}. \tag{6.2}$$

Elegant languages may be defined in such a way. E.g. in the Cyclops tensor framework the following C++-code may be used to execute the tensor contraction $Z_{abij} = V_{mnef} T_{efij} T_{abmn}$ [142]:

```
W["mnij"] = V["mnef"] * T["efij"];
Z["abij"] = W["mnij"] * T["abmn"];
```

Our syntax is very similar and briefly introduced in Section 6.1.1.

High level languages are common in tensor software. Some require providing summation indices explicitly [9, 48] and some use summation conventions [2, 114, 143]. In the latter group of papers, only Åhlander [2] points out potential ambiguities of Einstein notation and describes a formal set of rules to resolve ambiguities. We therefore review some of the examples in [2] in Section 6.1.2 to motivate the formal semantics in Section 6.1.3.

6.1.1 Syntax

YATeTo's DSL is embedded into Python. The basic building block is the `Tensor` class, in which the tensor's shape, the tensor's sparsity pattern, and the memory layout may be specified. `Tensor`'s do not appear directly in expressions, but objects of the class `IndexedTensor`. The latter objects are derived from a `Tensor` object via the `[.]` operator, and a string inside the square bracket labels each dimension of the tensor. Expressions are formed with the `*`, `+`, `-`, and the `<=` operator. The last operator is an assignment operator, as overloads for the `=` operator are illegal in Python.

The following example illustrates the language's syntax:

```
K = 4
L = 7
A = Tensor('A', (K, K))
B = Tensor('B', (K, L, L))
C = Tensor('C', (K, L))
D = Tensor('D', (L, K))
e = Tensor('e', (K,))
f = Tensor('f', (L,))
kernel = A['ij'] <= -A['ij'] + 2.0 * B['ilk'] * \
                     C['jl'] * (D['km'] * e['m'] - f['k'])
```

6.1.2 Einstein notation by example

The Einstein convention is sometimes problematic, as it might lead to ambiguous expressions. While the ambiguity may be resolved by a hu-

man reader, a computer language must be unambiguous. Consider the following examples [2]:

$$A_i = B_i C_i \tag{6.3}$$
$$A_i = B_{ii} \tag{6.4}$$
$$A_{ii} = B_{ii} \tag{6.5}$$
$$A = B_i C_i D_i E_i \tag{6.6}$$
$$A_i = B_i + 4 \tag{6.7}$$

In (6.3), it is unclear whether one shall sum over the right-hand side and then assign a scalar to each entry of the vector A, or if one should multiply B and C component-wise (the Hadamard product).

The source of the ambiguity is that there is no distinction between the left-hand side and right-hand side in an equality. In the DSL, due to the assignment operator `<=`, we always have a left-hand side and a right-hand side. We define that indices on the left-hand side are free indices for which no sum is implied. Therefore, (6.3) is unambiguously identified as Hadamard product. Summing the right-hand side may be achieved by rewriting the expression as following:

$$A_i := B_j C_j 1_i, \tag{6.8}$$

where 1_i is a vector of ones with the same shape as A_i.

In (6.4) and (6.5) the ambiguity may be resolved using the same convention, that is, i is bound by the left-hand side and is not summed over. However, repeated indices complicate the implementation. For instance, the expression

$$A_{ij} := B_{ik} C_{kkj} \tag{6.9}$$

can be interpreted as matrix-matrix multiplication, that is, the first two dimensions of C are regarded as row vector. But then, the rows of C have non-unit stride (in column-major order), and cannot be mapped to (standard) GEMM. In order to simplify the implementation, repeated indices are prohibited in YATeTo. Instead, one may rewrite (6.4), (6.5), and (6.9) with an appropriately sized identity matrix δ:

$$A_i := B_{ij} \delta_{ij} \tag{6.10}$$
$$A_{ij} := A_{ij} - A_{ij} \delta_{ij} + B_{ij} \delta_{ij} \tag{6.11}$$
$$A_{ij} := B_{ik} C_{klj} \delta_{lk} \tag{6.12}$$

Index i appears four times in (6.6), which is not covered by Einstein as his convention only considers indices appearing twice. Indices appearing multiple times are useful for DG methods, for example in quadrature rules:

$$\int_{-1}^{1} \phi(x) f(x) \, \mathrm{d}x \approx \sum_{i=1}^{N_q} \phi(x_i) f(x_i) w_i, \tag{6.13}$$

where w_i are quadrature weights and x_i are quadrature points. We therefore interpret products with indices appearing more than two times in the sense that indices are summed only once and the sum contains all product terms. A similar point of view is taken by Åhlander [2].

The last example, (6.7), questions whether one can add two tensors with different shape. The semantics of Åhlander [2] allow such statements: The scalar 4 is added component-wise to the vector B. Allowing such operations is likely confusing and non-intuitive in the author's opinion: In the standard definition of a vector space V, addition takes two vectors in V and returns a vector in V. Additions between elements of different vector spaces are undefined (e.g. adding a vector to a matrix), such that a language allowing to do so does not conform with the standard language used in linear algebra. The situation for products is different. Here, a user expects – unless one of the operands is a scalar – that a product is a "special" operation (e.g. matrix-vector product, dot product, n-mode product, Kronecker product, Khatri-Rao product, Hadamard product, etc. [86]). Therefore, we define that adding two tensors requires identical tensor shapes, and (6.7) must be rewritten to account for the broadcast of 4 explicitly:

$$A_i := B_i + 4 \cdot 1_i \tag{6.14}$$

6.1.3 Semantics

In Section 6.1.2 we have motivated the language interpretation by examples. Here, we formalise the language from the bottom up.

Definition 1. *An assignment statement is given by*

$$X[\alpha] := y.$$

Tensors are denoted with large letters (X), indices with Greek letters (α), and arithmetic expressions with small letters (y). An arithmetic expression is either

1. *an indexed tensor $Y[\beta]$,*

2. *a sum of* $p \geq 2$ *arithmetic expressions* $x_1 + \cdots + x_p$,

3. *or a product of* $q \geq 2$ *arithmetic expressions* $x_1 * \ldots * x_q$.

Indices play a fundamental role in the semantics of the language. We define properties and manipulation of indices in the following.

Definition 2. *We define indices as strings over an alphabet* Σ *(e.g.* $\Sigma = \{\mathtt{a}, \ldots, \mathtt{z}\}$*). Repeated indices are prohibited, such that for a valid index string* α *of length* n *we have*

$$\alpha \in \{w_1 \ldots w_n \in \Sigma^* : w_i \neq w_j \text{ whenever } i \neq j\}.$$

Index-strings $\alpha = \alpha_1 \ldots \alpha_n$ *can be converted to sets using* $\mathrm{set}(\alpha)$*, which is defined as*

$$\mathrm{set}(\alpha) = \mathrm{set}(\alpha_1 \ldots \alpha_n) = \bigcup_{i=1}^{n} \{\alpha_i\}.$$

An index set $\mathcal{B} \subset \Sigma$ *can also be converted to an index string using* $\mathrm{str}(\mathcal{B})$*. The* str *operation puts the characters in lexicographical order* ($\preceq$) *for uniqueness. That is, let* $\mathcal{B} = \{\beta_1, \ldots, \beta_m\}$ *with* $\beta_1 \preceq \ldots \preceq \beta_m$*, then*

$$\mathrm{str}(\mathcal{B}) = \beta_1 \ldots \beta_m$$

String projection is denoted with $\alpha \setminus \mathcal{B}$*, where* $\mathcal{B} \subset \Sigma$ *is a set of characters. That is,* $\alpha \setminus \mathcal{B}$ *removes all characters in* $\mathcal{B}$ *from* α*. Concatenation of* α *and* β *is denoted by* $\alpha\beta$*.*

The $\sqcup$*-operator is defined as a disjoint union of indices, i.e.*

$$\alpha \sqcup \beta := \alpha(\beta \setminus \mathrm{set}(\alpha)).$$

Lastly, every index in an index string has a size, given by the function

$$\mathrm{size} : \Sigma \to \mathbb{N}.$$

Based on the size function, the index string $\alpha_1 \ldots \alpha_n$ *spans an iteration space, defined as*

$$\mathbb{S}(\alpha) = \bigtimes_{k=1}^{n} \{0, 1, \ldots, \mathrm{size}(\alpha_k) - 1\}.$$

For the $*$-operation we require a formalism to map index strings to tensor dimensions. We therefore introduce projection and permutation functions, which is a common concept in tensor software [143, 144].

Definition 3. *A permutation and projection function is a function*

$$\pi[\beta|\alpha] : \mathbb{S}(\alpha) \to \mathbb{S}(\beta).$$

The input to such a function is a tuple $\boldsymbol{a} \in \mathbb{S}(\alpha)$ *(e.g.* $\boldsymbol{a} = (0, 1, 2)$*). The permutation and projection functions are uniquely determined by an input index string* α *and an output index string* $\beta = \beta_1 \dots \beta_n$ *in the following way:*

$$\pi[\beta|\alpha](\boldsymbol{a}) = \left(a_{\alpha.position(\beta_1)}, \dots, a_{\alpha.position(\beta_n)}\right), \tag{6.15}$$

where $\alpha.position(\beta_i)$ *returns at which position* β_i *was found in* α*. The projection and permutation function is undefined if* $\mathrm{set}(\beta) \not\subseteq \mathrm{set}(\alpha)$*.*

We also define the projection of iteration spaces as following ($\boldsymbol{b} \in \mathbb{S}(\beta)$*):*

$$\mathbb{P}[\beta|\alpha](\boldsymbol{b}) = \{\boldsymbol{a} \in \mathbb{S}(\alpha) : \pi[\beta|\alpha](\boldsymbol{a}) = \boldsymbol{b}\}. \tag{6.16}$$

The following example clarifies the concept of projection and permutation functions:

Example 1. *Consider the matrix-matrix product* `C['ij'] <= A['ik'] * B['kj']`*, where* $A \in \mathbb{R}^{I \times K}, B \in \mathbb{R}^{K \times J}$*, and* $C \in \mathbb{R}^{I \times J}$*. A valid mathematical representation of matrix multiplication is*

$$\forall (i, j) \in \mathbb{S}(\mathtt{ij}) : C_{(i,j)} := \sum_{(i',k',j') \in \{i\} \times [0..K-1] \times \{j\}} A_{(i',k')} B_{(k',j')}.$$

The indices of `A,B,C` *are given by* $\alpha :=$ `ik`*,* $\beta :=$ `kj`*,* $\gamma :=$ `ij`*, respectively. We define a combined index string* $\omega :=$ `ikj`*. The projection functions w.r.t.* ω *are given by* $\pi[\alpha|\omega](\boldsymbol{w}) = (w_1, w_2)$*,* $\pi[\beta|\omega](\boldsymbol{w}) = (w_2, w_3)$*,* $\pi[\gamma|\omega](\boldsymbol{w}) = (w_1, w_3)$*. Moreover, we compute the projection space*

$$\mathbb{P}[\gamma|\omega](\boldsymbol{c}) = \{\boldsymbol{w} \in \mathbb{S}(\omega) : (w_1, w_3) = (c_1, c_2)\} = \{c_1\} \times [0 \,..\, K-1] \times \{c_2\}.$$

The matrix multiplication example can now be formulated systematically:

$$\forall \boldsymbol{c} \in \mathbb{S}(\gamma) : C_{\boldsymbol{c}} := \sum_{\boldsymbol{w} \in \mathbb{P}[\gamma|\omega](\boldsymbol{c})} A_{\pi[\alpha|\omega](\boldsymbol{w})} B_{\pi[\beta|\omega](\boldsymbol{w})}.$$

For the `*`-operation we also need to analyse which indices are summation indices.

Definition 4. *The union function $\mathcal{U}$, which combines all indices appearing in an arithmetic expression, is defined as*

$$\begin{aligned}\mathcal{U}[X[\alpha]] &= \mathrm{set}(\alpha)\\ \mathcal{U}[x_1 + \ldots + x_p] &= \mathcal{U}[x_1] \cup \ldots \cup \mathcal{U}[x_p]\\ \mathcal{U}[x_1 * \ldots * x_p] &= \mathcal{U}[x_1] \cup \ldots \cup \mathcal{U}[x_p]\end{aligned}$$

The set of potential summation indices of the $*$-operation is given by:

$$\mathcal{S}[x_1 * \ldots * x_p] = \bigcup_{\substack{1\le i,j\le p\\ i\neq j}} \mathcal{U}[x_i] \cap \mathcal{U}[x_j]$$

We note that $\mathcal{S}[x_1 * \ldots * x_p]$ respects the summation convention, as every index that appears twice is a potential summation index.

The following definition formalises the semantics of the language.

Definition 5. *The semantics of an assignment statement $X[\alpha] := y$ is*

$$\forall \boldsymbol{a} \in \mathbb{S}(\alpha) : X_{\boldsymbol{a}} := [y|\alpha](\boldsymbol{a}).$$

We call the square bracket function the evaluation function, which depends on bound indices α and is the map

$$[y|\alpha] : \mathbb{S}(\alpha) \to \mathbb{R}\,.$$

The evaluation function for an indexed tensor is defined by

$$[Y[\beta]|\alpha](\boldsymbol{a}) = Y_{\pi[\beta|\alpha](\boldsymbol{a})}\,.$$

For the $+$-operation of arity $p \geq 2$, we define

$$[x_1 + \ldots + x_p|\alpha](\boldsymbol{a}) = [x_1|\alpha](\boldsymbol{a}) + \ldots + [x_p|\alpha](\boldsymbol{a}).$$

The $*$-operation of arity $q \geq 2$ is defined by

$$[x_1 * \ldots * x_q|\alpha](\boldsymbol{a}) = \sum_{\boldsymbol{s}\in\mathbb{P}[\alpha|\alpha\sqcup\sigma](\boldsymbol{a})} [x_1|\alpha \sqcup \sigma](\boldsymbol{s}) \cdot \ldots \cdot [x_q|\alpha \sqcup \sigma](\boldsymbol{s}),$$

where the set of summation indices is $\sigma = \mathrm{str}(\mathcal{S}[x_1 * \ldots * x_q])$.

The application of the semantic rules is demonstrated in the following example.

Example 2. *Consider the following expression*

$$A[\mathtt{ij}] := A[\mathtt{ij}] + \underbrace{B[\mathtt{ilk}] * C[\mathtt{jl}] * (\overbrace{D[\mathtt{kmi}] * E[\mathtt{mi}] + F[\mathtt{ik}]}^{=:y_2})}_{=:y_1}.$$

The assignment rule gives

$$\forall \boldsymbol{a} \in [0\,..\,I-1] \times [0\,..\,J-1] : X_{\boldsymbol{a}} := [A[\mathtt{ij}] + y_1|\mathtt{ij}](\boldsymbol{a})$$

Applying the addition rule yields

$$[A[\mathtt{ij}] + y_1|\mathtt{ij}](\boldsymbol{a}) = A_{\boldsymbol{a}} + [y_1|\mathtt{ij}](\boldsymbol{a}).$$

The evaluation of y_1 is given by

$$[y_1|\mathtt{ij}](\boldsymbol{a}) = [B[\mathtt{ilk}] * C[\mathtt{jl}] * y_2|\mathtt{ij}](\boldsymbol{a}) = \sum_{\boldsymbol{i} \in \mathbb{I}} B_{(i_1,i_4,i_3)} C_{(i_2,i_4)} [y_2|\mathtt{ijkl}](\boldsymbol{i}),$$

where $\mathbb{I} = \{a_1\} \times \{a_2\} \times [0\,..\,K-1] \times [0\,..\,L-1]$. Note that the sum is taken over $\mathtt{k}$ and $\mathtt{l}$ but not over $\mathtt{m}$, as it is bound in y_2, and not over $\mathtt{i}$, as it is in the set of bound indices. The evaluation of y_2 yields

$$\begin{aligned}[y_2|\mathtt{ijkl}](\boldsymbol{i}) &= [D[\mathtt{kmi}] * E[\mathtt{mi}] + F[\mathtt{ik}]|\mathtt{ijkl}](\boldsymbol{i}) \\ &= \left(\sum_{\boldsymbol{j} \in \mathbb{J}} D_{(j_1,j_3,j_5)} E_{(j_5,j_1)} \right) + F_{(i_1,i_3)}\,,\end{aligned}$$

where $\mathbb{J} = \{i_1\} \times \{i_2\} \times \{i_3\} \times \{i_4\} \times [0\,..\,M-1]$. Note that indices $\mathtt{i}$ and $\mathtt{k}$ are not summed, as they are bound beforehand.

Finally, we impose the following restrictions on expressions.

Definition 6. *The result of an operation is a (virtual) tensor, whose set of indices is determined with*

$$\begin{aligned}\mathrm{Ix}[Y[\beta]|\mathcal{B}] &= \mathrm{set}(\beta), \\ \mathrm{Ix}[x_1 + \ldots + x_p|\mathcal{B}] &= \bigcup_{i=1}^{p} \mathrm{Ix}[x_i|\mathcal{B}], \\ \mathrm{Ix}[x_1 * \ldots * x_p|\mathcal{B}] &= \left(\bigcup_{i=1}^{p} \mathrm{Ix}[x_i|\mathcal{I}] \right) \setminus \mathcal{I} \text{ with } \mathcal{I} = \mathcal{S}[x_1 * \ldots * x_p] \cup \mathcal{B}.\end{aligned}$$

An expression is valid if and only if

1. *there are no undefined permutation and projection functions (see Definition 3),*
2. *the indices of summands in a* `+` *operation are equal up to permutation, that is,* $\forall 1 \leq i \leq p : \mathrm{Ix}[x_1 + \ldots + x_p | \mathcal{B}] = \mathrm{Ix}[x_i | \mathcal{B}]$,
3. *the indices in an assignment statement* $X[\alpha] := y$ *match, that is,* $\mathrm{set}(\alpha) = \mathrm{Ix}[y | \mathrm{set}(\alpha)]$.

6.2 Optimisation pipeline

An expression formed in the DSL generates an Abstract Syntax Tree (AST). The raw AST is hardly useful for generating efficient code, hence a sequence of optimisation steps is required. In order to maintain a modular and flexible code base, the visitor pattern is chosen as the main building block. The visitor pattern is effective whenever many algorithms shall be implemented and seldom changes of the underlying data structure are expected [54].

The first visitor, `Deduce Indices`, implements the semantic rules introduced in Section 6.1.3. Next, the number of operations is reduced in `Equivalent Sparsity Pattern` and `Strength Reduction`. The former step reduces unnecessary multiplications with zeros in tensor contractions, which is to the author's best knowledge a novel algorithm contributed in this thesis. The latter optimisation step, strength reduction [95], is a standard optimisation step in which a product of more than two operands is mapped to a binary tree with minimal number of floating point operations. The output of strength reduction is a binary tree containing `Product` and `IndexSum` nodes [95], which are mapped to contractions (i.e. a product followed by several index sums) in the `Find Contractions` visitor. Next, the memory layout of each tensor is fixed in `Compute Memory Layout`, which is a prerequisite for the subsequent `Find Index Permutations`. The latter is a dynamic programming algorithm which permutes intermediate results (i.e. temporary tensors) in order to minimise the execution time according to a heuristic cost function. The next two steps are only of technical nature, and in the last two steps a rudimentary support for tensor prefetching is introduced.

In this section, optimisation algorithms acting on the AST are described, see the left box in Figure 9. The transition from an AST to code is presented in Section 6.3.

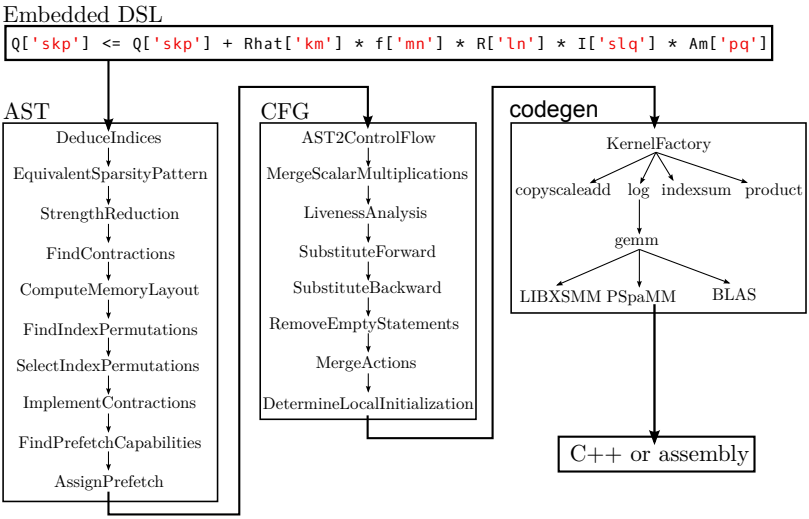


Figure 9: The figure shows the optimisation pipeline of YATeTo [157]. All optimisation steps are implemented using the visitor pattern in the Abstract Syntax Tree (AST) and Control Flow Graph (CFG) stages, and using the factory pattern on the code generation stage. The output is C++-code or inline assembly.

6.2.1 Equivalent sparsity patterns

A sparse tensor is a tensor where a subset of entries is zero. If the sparsity pattern of a tensor is known (i.e. the locations of the zero-entries), then an implementation might exploit the sparsity pattern, as in the following example:

$$\begin{pmatrix} K_{11} & 0 & K_{13} \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} Q_{11} & 0 & Q_{13} \\ 0 & Q_{22} & 0 \\ Q_{31} & 0 & 0 \end{pmatrix} = \begin{pmatrix} K_{11}Q_{11} + K_{13}Q_{31} & 0 & K_{11}Q_{13} \\ 0 & 0 & 0 \end{pmatrix}. \tag{6.17}$$

Here, one requires only 4 operations instead of the 30 operations which would be required if two dense matrices of the same size are multiplied. In particular, the result is independent of the entry Q_{22}.

Consider now a third matrix being right-multiplied to (6.17):

$$\begin{pmatrix} K_{11} & 0 & K_{13} \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} Q_{11} & 0 & Q_{13} \\ 0 & Q_{22} & 0 \\ Q_{31} & 0 & 0 \end{pmatrix} \begin{pmatrix} 0 & 0 \\ A_{21} & 0 \\ 0 & A_{32} \end{pmatrix} = \begin{pmatrix} 0 & K_{11}Q_{13}A_{32} \\ 0 & 0 \end{pmatrix}. \tag{6.18}$$

The operation count reduces from 4 operations to 2 operations, that is, the operation count of the above matrix chain product of matrices is less than the operation count of (6.17).

One cannot achieve the optimal operation count by only considering sequences of pairwise sparse matrix matrix multiplication in the above example. That is, computing $t_1 := KQ$ followed by $t_1 A$ requires 5 operations and computing $t_2 := QA$ followed by Kt_2 requires 3 operations. However, one may mask the irrelevant entries beforehand:

$$\begin{pmatrix} K_{11} & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} 0 & 0 & Q_{13} \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & A_{32} \end{pmatrix} = \begin{pmatrix} 0 & K_{11}Q_{13}A_{32} \\ 0 & 0 \end{pmatrix}. \tag{6.19}$$

Pairwise sparse matrix matrix multiplication on the masked matrix chain product gives the optimal operation count of 2.

Ideally, we would adjust the tensors in (6.18) according to (6.19), such that we can apply binary matrix matrix multiplication with optimal operation count. The latter is not possible in general, because a tensor might be required in several product expressions, where the irrelevant entries in one product expression become relevant in another product expression. Therefore, YATeTo computes so-called equivalent sparsity patterns (EQSPPs) for each product expression. E.g. for (6.18) the EQSPPs are given by the Boolean tensors

$$\hat{K} := \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \quad \hat{Q} := \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \quad \hat{A} := \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 1 \end{pmatrix}. \tag{6.20}$$

The EQSPPs might then be used in the code generation stage for (6.18), e.g. by adjusting loop ranges in the matrix matrix multiplication code, such that irrelevant entries are never loaded and multiplied. We emphasise, however, that YATeTo only stores a single representation of a tensor in memory, i.e. one which covers all non-zeros of a tensor.

The remainder of this section is dedicated to the automatic computation of EQSPPs for general tensor contractions. We note that we make a *no*

cancellation assumption [26], as it is common when dealing with sparsity. That is, we assume that the inner product of two vectors is never zero (which is quite unlikely, because the inner product of two d-dimensional vectors is only zero on a hyperplane of dimension $d-1$).

Formal problem statement

We consider product expressions of the form

$$U[\alpha] := T^1[\beta_1] * \ldots * T^q[\beta_q]. \tag{6.21}$$

We are looking for entries in each tensor which do not affect the result of above operation. Relevant and irrelevant entries of tensor T^i are represented by a Boolean tensor $\hat{\Theta}^i$ of the same shape. If we cannot remove anymore entries in $\hat{\Theta}^i$ (i.e. set them to 0) without changing the result of (6.21), then we call it an equivalent sparsity pattern.

In the following EQSPPs are defined precisely:

Definition 7 (Equivalent sparsity patterns [157])**.** *Let a generic product expression as in* (6.21) *be given, with the following semantics:*

$$\forall \boldsymbol{a} \in \mathbb{S}(\alpha) : U_{\boldsymbol{a}} = \sum_{\boldsymbol{j} \in \mathbb{P}[\alpha|\alpha \sqcup \sigma](\boldsymbol{a})} T^1_{\pi[\beta_1|\alpha\sqcup\sigma](\boldsymbol{j})} \cdot \ldots \cdot T^q_{\pi[\beta_q|\alpha\sqcup\sigma](\boldsymbol{j})},$$

where $\sigma = \mathrm{str}(\mathcal{S}[T^1[\beta_1] * \ldots * T^n[\beta_q]])$.

We call $\hat{\Theta}^k$, $k = 1, \ldots, n$, *equivalent sparsity patterns w.r.t.* (6.21) *if*

1. $\hat{\Theta}^k$ *is a Boolean tensor with the same shape as* T^k, *i.e.*

$$\forall \boldsymbol{b} \in \mathbb{S}(\beta^k) : \hat{\Theta}^k_{\boldsymbol{b}} \in \{0, 1\}.$$

2. *Masking the tensors* T^k *with* $\hat{\Theta}^k$ *leaves the result of* (6.21) *unchanged. That is,* $\forall \boldsymbol{a} \in \mathbb{S}(\alpha) : U_{\boldsymbol{a}} = \hat{U}_{\boldsymbol{a}}$, *where*

$$\forall \boldsymbol{a} \in \mathbb{S}(\alpha) : \hat{U}_{\boldsymbol{a}} = \sum_{\boldsymbol{j} \in \mathbb{P}[\alpha|\alpha \sqcup \sigma](\boldsymbol{a})} \hat{T}^1_{\pi[\beta_1|\alpha\sqcup\sigma](\boldsymbol{j})} \cdot \ldots \cdot \hat{T}^q_{\pi[\beta_q|\alpha\sqcup\sigma](\boldsymbol{j})},$$

and

$$\forall \boldsymbol{b} \in \mathbb{S}(\beta_k) : \hat{T}^k_{\boldsymbol{b}} = \begin{cases} T^k_{\boldsymbol{b}} & \text{if} \quad \hat{\Theta}^k_{\boldsymbol{b}} = 1, \\ 0 & \text{if} \quad \hat{\Theta}^k_{\boldsymbol{b}} = 0. \end{cases}$$

3. *The number of non-zeros of* $\hat{\Theta}^k$, $k = 1, \ldots, n$, *is minimal, that is, we cannot set a non-zero to zero without implying*

$$\exists \boldsymbol{a} \in \mathbb{S}(\alpha) : \hat{U}_{\boldsymbol{a}} \neq U_{\boldsymbol{a}}\,.$$

Computation of EQSPPs

A product expression as in (6.21) can be regarded as an (potentially huge) outer product of all tensors followed by a sum reduction. The following lemma shows that we only need the outer product for computing EQSPPs.

Lemma 3 ([157])**.** *The EQSPPs w.r.t.* (6.21) *are equivalent to the EQSPPs w.r.t.*

$$Z[\alpha \sqcup \sigma] := T^1[\beta_1] * \ldots * T^q[\beta_q].$$

Proof. We simply check if the EQSPPs $\hat{\Theta}^k$ for Z fulfil the three conditions in Definition 7 for U:

1. Fulfilled trivially.

2. The entries of Z are given by

$$\forall \boldsymbol{j} \in \mathbb{S}(\alpha \sqcup \sigma) : Z_{\boldsymbol{j}} = T^1_{\pi[\beta_1|\alpha\sqcup\sigma](\boldsymbol{j})} \cdot \ldots \cdot T^q_{\pi[\beta_q|\alpha\sqcup\sigma](\boldsymbol{j})},$$

   where we used that $\mathbb{P}[\alpha \sqcup \sigma|\alpha \sqcup \sigma](\boldsymbol{w}) = \{\boldsymbol{w}\}$ and the sum sign over a single element can be dropped. Masking the tensors T^k with $\hat{\Theta}^k$ leaves the result U unchanged, because

$$\forall \boldsymbol{a} \in \mathbb{S}(\alpha) : U_{\boldsymbol{a}} = \sum_{\boldsymbol{j} \in \mathbb{P}[\alpha|\alpha\sqcup\sigma](\boldsymbol{a})} Z_{\boldsymbol{j}} = \sum_{\boldsymbol{j} \in \mathbb{P}[\alpha|\alpha\sqcup\sigma](\boldsymbol{a})} \hat{Z}_{\boldsymbol{j}} = \hat{U}_{\boldsymbol{a}}\,.$$

3. Assume there exists $\bar{\Theta}^k$ with less non-zeros than $\hat{\Theta}^k$. Then there exists an $\boldsymbol{f} \in \mathbb{S}(\alpha \sqcup \sigma)$ such that $Z_{\boldsymbol{f}} = \hat{Z}_{\boldsymbol{f}} \neq \bar{Z}_{\boldsymbol{f}}$, because otherwise $\hat{\Theta}^k$ would not be minimal for Z. As $Z_{\boldsymbol{f}}$ does not get cancelled in sums, it follows that there exists an index $\boldsymbol{g} \in \mathbb{S}(\alpha)$ where $\bar{U}_{\boldsymbol{g}} \neq U_{\boldsymbol{g}}$.

□

The question remains how to compute the EQSPPs for the outer product expression. The key ingredient here is to realise that Z contains all possible products of tensor entries. Hence, in order to check if a particular entry $T^k_{\boldsymbol{b}}$, for $\boldsymbol{b} \in \mathbb{S}(\beta_k)$, is irrelevant, one has to check whether it is

involved in any non-zero product. In other words, if $\forall \boldsymbol{j} \in \mathbb{P}[\beta_k|\alpha \sqcup \sigma](\boldsymbol{b}) : Z_{\boldsymbol{j}} = 0$, then $T^k_{\boldsymbol{b}}$ is irrelevant.

One does not need to compute Z explicitly. Instead one may recast the line of thought in the last paragraph as Boolean tensor contractions:

Theorem 2 ([157])**.** *The EQSPPs w.r.t.* (6.21) *are given by*

$$\forall \boldsymbol{b} \in \mathbb{S}(\beta_k) : \hat{\Theta}^k_{\boldsymbol{b}} := \sum_{\boldsymbol{l} \in \mathbb{P}[\beta_k|\alpha \sqcup \sigma](\boldsymbol{b})} \Theta^1_{\pi[\beta_1|\alpha \sqcup \sigma](\boldsymbol{l})} \cdot \ldots \cdot \Theta^q_{\pi[\beta_q|\alpha \sqcup \sigma](\boldsymbol{l})}, \quad (6.22)$$

where Θ^l is the sparsity pattern of T^l, $l = 1, \ldots, q$. The $+$ and $\cdot$ operations are identified with the $\vee$ and $\wedge$ operations, respectively.

Proof. We are going to show that (6.22) computes the EQSPPs for the product Z of tensors $T^1, \ldots, T^q$ (see Lemma 3).

Condition 1 is satisfied as we interpret addition and multiplication with logical operations (i.e. $x + y \equiv x \vee y$ and $x \cdot y \equiv x \wedge y$). Hence, $\hat{\Theta}^k$ is a Boolean tensor.

In order to satisfy condition 2, we need to show that $Z = \hat{Z}$. The latter is true when the sparsity pattern of Z, say ζ, is equivalent to the sparsity pattern of $\hat{Z}$, say $\hat{\zeta}$. The tensor ζ is given by

$$\forall \boldsymbol{j} \in \mathbb{S}(\alpha \sqcup \sigma) : \zeta_{\boldsymbol{j}} = \Theta^1_{\pi[\beta_1|\alpha \sqcup \sigma](\boldsymbol{j})} \cdot \ldots \cdot \Theta^q_{\pi[\beta_q|\alpha \sqcup \sigma](\boldsymbol{j})}, \quad (6.23)$$

and $\hat{\zeta}$ is given equivalently by putting a hat on every tensor in above equation. We first derive the following useful identity from (6.22) and (6.23):

$$\forall \boldsymbol{b} \in \mathbb{S}(\beta_k) : \hat{\Theta}^k_{\boldsymbol{b}} = \sum_{\boldsymbol{l} \in \mathbb{P}[\beta_k|\alpha \sqcup \sigma](\boldsymbol{b})} \zeta_{\boldsymbol{l}} = \Theta^k_{\boldsymbol{b}} \cdot \sum_{\boldsymbol{l} \in \mathbb{P}[\beta_k|\alpha \sqcup \sigma](\boldsymbol{b})} \zeta_{\boldsymbol{l}}. \quad (6.24)$$

The latter equality follows from idempotence and $\pi[\beta_k|\alpha \sqcup \sigma](\boldsymbol{l}) = \boldsymbol{b}$ if $\boldsymbol{l} \in \mathbb{P}[\beta_k|\alpha \sqcup \sigma](\boldsymbol{b})$ (and thus $\Theta^k_{\pi[\beta_k|\alpha \sqcup \sigma](\boldsymbol{l})} = \Theta^k_{\boldsymbol{b}}$). Plugging (6.24) into the computation of $\hat{\zeta}$ gives

$$\begin{aligned} &\forall \boldsymbol{j} \in \mathbb{S}(\alpha \sqcup \sigma) : \hat{\zeta}_{\boldsymbol{j}} = \hat{\Theta}^1_{\pi[\beta_1|\alpha \sqcup \sigma](\boldsymbol{j})} \cdot \ldots \cdot \hat{\Theta}^q_{\pi[\beta_q|\alpha \sqcup \sigma](\boldsymbol{j})} \\ &= \zeta_{\boldsymbol{j}} \cdot \left(\sum_{\boldsymbol{l} \in \mathbb{P}[\beta_1|\alpha \sqcup \sigma](\pi[\beta_1|\alpha \sqcup \sigma](\boldsymbol{j}))} \zeta_{\boldsymbol{l}} \right) \cdot \ldots \cdot \left(\sum_{\boldsymbol{l} \in \mathbb{P}[\beta_q|\alpha \sqcup \sigma](\pi[\beta_q|\alpha \sqcup \sigma](\boldsymbol{j}))} \zeta_{\boldsymbol{l}} \right). \end{aligned} \quad (6.25)$$

Note that $\boldsymbol{j} \in \mathbb{P}[\beta_k|\alpha \sqcup \sigma](\pi[\beta_k|\alpha \sqcup \sigma](\boldsymbol{j}))$. Therefore, every sum in above equation includes $\zeta_{\boldsymbol{j}}$. By the absorption law ($x \wedge (x \vee y) = x$) it follows

$$\forall \boldsymbol{j} \in \mathbb{S}(\alpha \sqcup \sigma) : \hat{\zeta}_{\boldsymbol{j}} = \zeta_{\boldsymbol{j}}.$$

To show that condition 3 holds, assume that there exists another set of $\bar{\Theta}^k$, with fewer non-zeros than $\hat{\Theta}^k$. Thus, for some $m \in [1\,..\,q]$ there must exist an entry $\boldsymbol{b} \in \mathbb{S}(\beta_m)$ such that $\bar{\Theta}^m_{\boldsymbol{b}} = 0$ and $\hat{\Theta}^m_{\boldsymbol{b}} = 1$. For the index $\boldsymbol{b}$ it holds that $\sum_{\boldsymbol{l} \in \mathbb{P}[\beta_m|\alpha \sqcup \sigma](\boldsymbol{b})} \zeta_{\boldsymbol{l}} = 1$ because otherwise $\hat{\Theta}^m_{\boldsymbol{b}} \neq 1$ would follow from (6.24). Therefore, there exists an index $\boldsymbol{c} \in \mathbb{P}[\beta_m|\alpha \sqcup \sigma](\boldsymbol{b})$ such that $\zeta_{\boldsymbol{c}} = 1$. As $\pi[\beta_m|\alpha \sqcup \sigma](\boldsymbol{c}) = \boldsymbol{b}$ we have

$$\bar{\zeta}_{\boldsymbol{c}} = \bar{\Theta}^1_{\pi[\beta_1|\alpha \sqcup \sigma](\boldsymbol{c})} \cdot \ldots \cdot \bar{\Theta}^m_{\boldsymbol{b}} \cdot \ldots \cdot \bar{\Theta}^q_{\pi[\beta_q|\alpha \sqcup \sigma](\boldsymbol{c})} = 0 \neq \zeta_{\boldsymbol{c}} = 1.$$

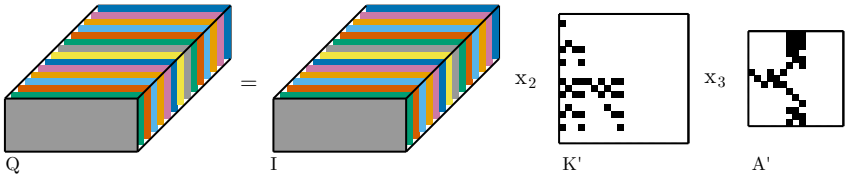
Therefore the set of sparsity patterns $\bar{\Theta}^k$ violates condition 2 and there cannot be another set of sparsity patterns with less non-zeros than $\hat{\Theta}^k$. □

Discussion

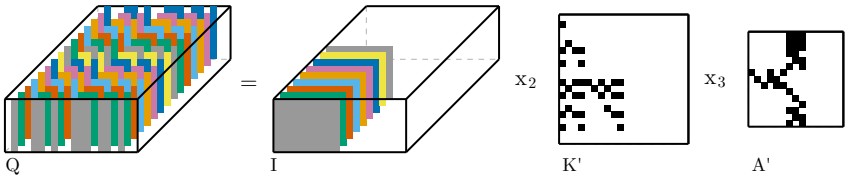
From Theorem 2 we know that the computation of EQSPPs can be cast as Boolean tensor contractions, using the language constructs developed for YATeTo. An implementation is therefore straightforward. Moreover, the outer product tensor Z does not need to be built explicitly and we can use strength reduction (see Section 6.2.2) to reduce the cost of computing EQSPPs.

Computing EQSPPs with Theorem 2 is likely more expensive than the original tensor contraction. Thus the method proposed here is only beneficial when the same tensor contraction is required multiple times. The latter is the case for our target applications because we expect to call the same kernels millions of times, and thus the cost of computing EQSPPs is easily amortised.

An example application of EQSPPs is shown in Figure 10. The example is taken from [156], where we added an additional dimension to the degrees of freedom tensor. The additional dimension showcases ensemble simulations, which are further discussed in Section 7.2.



(a) Tensor contraction with sparse tensors.



(b) Tensor contraction masked with equivalent sparsity patterns.

Figure 10: The figure shows the kernel $Q = I \times_2 K^T \times_3 A^T$, that is, $Q_{skp} := I_{slq} K_{lk} A_{qp}$. The tensor contraction in Figure 10b gives the same result as the tensor contraction in Figure 10a, but requires less floating point operations. The equivalent sparsity pattern algorithm automatically detects unnecessary non-zeros in tensors. Figure from [157].

6.2.2 Strength reduction

We consider again product expressions as in Equation (6.21). For example, consider the product

$$C[\mathtt{ij}] := A[\mathtt{lj}] * B[\mathtt{ikl}] * w[\mathtt{k}], \tag{6.26}$$

where all indices have size N. A naive implementation of such a product expression is given by the following code (assuming that the C-tensor is initially zero)

```
for (int i = 0; i < N; ++i)
  for (int j = 0; j < N; ++j)
    for (int l = 0; l < N; ++l)
      for (int k = 0; k < N; ++k)
        C[i + j*N] += A[l + j*N] * B[i + k*N + l*N*N] * w[k];
```

The innermost statement requires two multiplications and one addition, thus the total number of floating point operations is given by $3N^4$. One

easily identifies that the implementation should be separated in two steps in order to reduce the number of floating point operations:

```
for (int i = 0; i < N; ++i)
  for (int l = 0; l < N; ++l)
    for (int k = 0; k < N; ++k)
      Tmp[i + l*N] += B[i + k*N + l*N*N] * w[k];
for (int i = 0; i < N; ++i)
  for (int j = 0; j < N; ++j)
    for (int l = 0; l < N; ++l)
      C[i + j*N] += A[l + j*N] * Tmp[i + l*N];
```

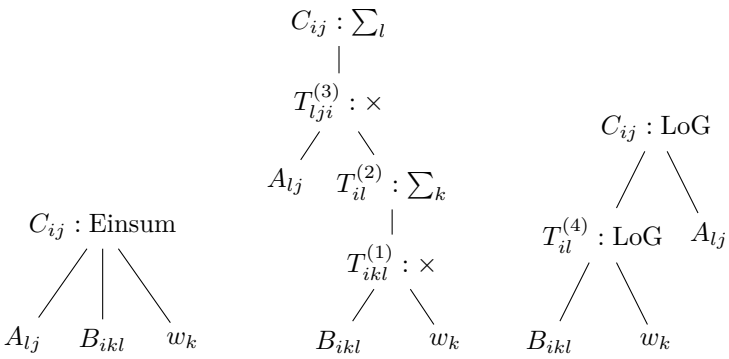
The number of operations is only $4N^3$ in above implementation. In the same manner, one could first contract A and B, however then the operation count is $2N^4 + 2N^3$.

But what is the optimal sequence of computations for general product expressions? The latter problem is well-known and is often called *strength reduction* in tensor literature. Strength reduction is first discussed by Lam et al. [97]: The authors first introduce two types of formulae, a multiplication formula and a summation formula. The multiplication formula $V[\dots] = X[\dots] \times Y[\dots]$, where the dots symbolise an index list, computes the "outer product" of two tensors (in the same way as the outer product is computed in Lemma 3). The summation formula $W[\dots] = \sum_i Z[\dots]$ returns the sum over a single index. Then, they define the operation minimisation problem as following: Given a general product expression, find the sequence of multiplication and summation formulae which minimise the operation count.

Lam et al. [97] show that the latter optimisation problem is NP-complete. However, if the number of terms in a product expression is small enough, an exhaustive search procedure is feasible. Lam et al. [97] propose smart exhaustive search procedures.

The number of terms in product expressions is expected to be small enough in DG methods, thus there is no need for novel algorithms. In YATeTo the original search procedure of [97] is implemented. For example, the output of the strength reduction for (6.26) is shown in Figure 11b.

The operation count in YATeTo is based on equivalent sparsity patterns. That is, the cost of a multiplication formula $V[\dots] = X[\dots] \times Y[\dots]$ is given by the number of non-zeros in V, where the sparsity pattern of V is computed as the outer product of the equivalent sparsity patterns of X and Y. Similarly, the cost of a summation formula $W[\dots] = \sum_i Z[\dots]$ is given by the number of non-zeros in Z minus the number of non-zeros in W.



(a) Input from DSL. **(b)** Strength reduction. **(c)** Loop-over-GEMM.

Figure 11: On the left, the input from the DSL is shown which contains a product expression with three terms. In the strength reduction step (middle), the initial AST is mapped to a binary which requires the minimum amount of floating point operations. Lastly, the AST is mapped to Loop-over-GEMM. Here, index permutations of intermediate results are chosen to minimise the cost function in Section 6.2.5. Figure from [157].

Sparsity is dealt with explicitly as we assume that the inputs to YATeTo are small tensors. If tensors are large, computing the operation count might become infeasible, as the complete sparsity pattern needs to be computed. However, the cost estimator is implemented as a visitor and one only needs to specify how to compute the cost of a multiplication and summation formula. Thus, one might replace the cost estimator, for example with one that only estimates intermediate sparsity patterns [96]. Furthermore, sparsity patterns itself are abstract in YATeTo. For example, the "sparsity pattern" of a dense tensor is only stored implicitly by storing the shape of the tensor. As such, YATeTo may deal with large dense tensors out of the box.

We note that extensions for the strength reduction algorithm [97] exist that minimise the operation count further by eliminating common subexpressions [63, 96]. These extensions are not implemented in YATeTo but could be included in future work.

6.2.3 Memory layouts

Moving further down the optimisation pipeline, we need to take account of the data structure of tensors in computer memory. Tensors are multi-dimensional objects but computer memory is one-dimensional. Thus, one requires a one-to-one relation between the multi-dimensional index of a tensor entry and the location in memory where the entry is stored. More specifically, we only need a memory location for every non-zero of a tensor. In the following, we call the bijection which maps every non-zero of a tensor to a memory location the memory layout of a tensor.

Bounding box memory layout

Classically, multi-dimensional arrays are either organised in row-major order (C order) or in column-major order (Fortran order), where the latter is used throughout this thesis. The (packed) column-major memory layout for a tensor $A \in \mathbb{R}^{n_1 \times n_2 \times \ldots \times n_d}$ is given by

$$(i_1, \ldots, i_d) \mapsto \sum_{k=1}^{d} i_k s_k, \tag{6.27}$$

where the strides s_k are defined as $s_k = \prod_{l=0}^{k-1} n_l$ with $n_0 := 1$. Note that we start counting at zero, i.e. $i_k \in [0\,..\,n_k - 1]$.

A simple extension of the column-major memory layout is the bounding box memory layout. The latter is useful whenever the tensor A is only non-zero within the index set $[b_1\,..\,B_1 - 1] \times \cdots \times [b_d\,..\,B_d - 1]$, where $0 \leq b_k < B_k \leq n_k$. We define it as

$$(i_1, \ldots, i_d) \mapsto \sum_{k=1}^{d} (i_k - b_k) t_k, \tag{6.28}$$

where $t_k = \prod_{l=0}^{k-1} (B_l - b_l)$.

Lastly, in some situations it might be beneficial to add artificial zeros to the leading dimension, so-called zero-padding, which can be used to avoid cache-line splits on some architectures [66]. The number of zeros that needs to be added is hardware dependent. E.g. on the Knights Landing architecture one would require that the size of the leading dimension is a multiple of $v = 8$ in double precision and $v = 16$ in single precision. Therefore, the initial interval $[b_1\,..\,B_1 - 1]$ may be adjusted [156]:

$$[b_1'\,..\,B_1' - 1] := [(b_1 - b_1 \bmod v)\,..\,(B_1 - 1 + (v - B_1 \bmod v) \bmod v)]. \tag{6.29}$$

Sparse memory layouts

In previous work on SeisSol, the Compressed Sparse Column (CSC) format was used for some matrices. The performance results for SeisSol indicate that the speed-up gained by using the CSC format instead of the dense format is rather mediocre (10–15 %) on the Sandy Bridge and Haswell architectures [16, 156] and almost non-existent on the Knights Landing architecture [64]. A major problem here is the efficient vectorisation on sparse memory layouts.

A notable exception are dense × sparse matrix-matrix multiplications, which can be implemented quite efficiently by hardwiring the sparsity pattern in the matrix multiplication kernel [14, 19, 20]. However, a problem here is that code is also data and as such the size of the L1i cache limits the number of non-zeros in the sparse matrix [14].

In light of these results, only a limited support for the CSC format is implemented in YATeTo.

6.2.4 Loop-over-GEMM

Tensor contractions shall be mapped to highly efficient GEMMs. In order to do so, the Loop-over-GEMM (LoG) approach is employed. LoG is systematically developed in [35, 101, 139, 144], therefore we only illustrate the basic ideas here.

We first recall the possible GEMM operations for real matrices [36]:

$$\begin{aligned} C &:= \alpha AB + \beta C, \quad C := \alpha A^T B + \beta C, \\ C &:= \alpha AB^T + \beta C, \quad C := \alpha A^T B^T + \beta C. \end{aligned} \tag{6.30}$$

C is a $M \times N$ matrix and α and β are scalars. The matrix A is a $M \times K$ matrix in normal mode or a $K \times M$ matrix in transposed mode. Similarly, B is $K \times N$ or $N \times K$ if transposed. The memory layout of C is given by

$$(m, n) \mapsto m + n \cdot LDC. \tag{6.31}$$

LDC is called the stride and must be greater than or equal M. The memory layout of A is given by

$$\begin{aligned} (m, k) &\mapsto m + k \cdot LDA \quad (\text{normal}, LDA \geq M), \\ (k, m) &\mapsto m + k \cdot LDA \quad (\text{transposed}, LDA \geq K), \end{aligned} \tag{6.32}$$

and similarly for B. Note that the right-hand side of the memory layout does not change, because A^T does not physically change the data organisation but only the way data is accessed.

The most important ingredient of the LoG-approach is that we may view a chunk of memory as a tensor or as a matrix without copying any data. For example, the memory layout of the tensor $\widehat{C}_{abgcd}$, where all indices are of size $S \in \mathbb{N}$, is given by

$$(a, b, g, c, d) \mapsto a + bS + gS^2 + cS^3 + dS^4. \tag{6.33}$$

We may view abg as the rows and cd as the columns of a matrix by constructing the bijection

$$(a + bS + gS^2) + (c + dS) \cdot S^3 \mapsto (m(a, b, g), n(c, d)) \tag{6.34}$$

where $m \in [0\,..\,S^3 - 1]$ and $n \in [0\,..\,S^2 - 1]$ and $m = a + bS + gS^2$ and $n = c + dS$. To show that the latter is really a bijection, we construct the inverse relation as following:

$$(m, n) \mapsto (a(m) + b(m)S + g(m)S^2) + (c(n) + d(n)S) \cdot S^3, \tag{6.35}$$

where

$$\begin{aligned} a(m) &= m \bmod S, \quad b(m) = \left\lfloor \tfrac{m}{S} \right\rfloor \bmod S, \quad g(m) = \left\lfloor \tfrac{m}{S^2} \right\rfloor \bmod S, \\ c(n) &= n \bmod S, \qquad d(n) = \left\lfloor \tfrac{n}{S} \right\rfloor \bmod S. \end{aligned} \tag{6.36}$$

The stride passed to GEMM may be larger than the number of rows. Therefore, we can also view a sub-tensor as matrix. For example, we may consider the index g fixed and only view the remaining entries as matrix:

$$gS^2 + (a + bS) + (c + dS) \cdot S^3 \mapsto (m(a, b), n(c, d)) \tag{6.37}$$

The term gS^2 is interpreted as constant offset (that can be included in a call to GEMM by adjusting the pointer to the first element). The index m runs from 0 to $S^2 - 1$ and $LDC = S^3$.

We now map an example tensor contraction to Loop-over-GEMM:

$$\widehat{C}_{abgcd} := \widehat{C}_{abgcd} + \sum_{m,n} \widehat{A}_{cdfeg} \widehat{B}_{abegf}. \tag{6.38}$$

Following Shi et al. [139], we fuse indices with $(\dots)$, fix a single index with [.] and denote transposes with the superscript T. First of all, note

that the index g appears in all three terms, hence it cannot be part of a GEMM and we fix it with $[g]$. The indices ab appear in $\widehat{B}$ and $\widehat{C}$ in the same order and cd appear in $\widehat{A}$ in $\widehat{C}$ in the same order. Therefore we take the fused indices (ab) as "m" index and the fused indices (cd) as "n" index. The indices e and f are not adjacent in $\widehat{B}$ and we therefore cannot fuse those. We may take either e or f as "k" index and fix the other. We obtain

$$\widehat{C}_{(ab)[g](cd)} := \widehat{C}_{(ab)[g](cd)} + \widehat{B}_{(ab)[e][g]f} \widehat{A}^T_{(cd)f[e][g]}. \tag{6.39}$$

The indices e and g are fixed and we therefore loop over them. Note that we switched $\widehat{A}$ and $\widehat{B}$ as the "m"-index has to appear in the left-hand term. Moreover, $\widehat{A}$ is transposed as the "k"-index and "n"-index are flipped. Listing 1 shows a possible implementation in the C-language.

```
for (int _g = 0; _g < 8; ++_g) {
  double const* _A = Bhat + 512*_g;
  double const* _B = Ahat + 4096*_g;
  double * _C = Chat + 64*_g;
  for (int _e = 0; _e < 8; ++_e) {
    double const* _Ain = _A + 64*_e;
    double const* _Bin = _B + 512*_e;
    double * _Cin = _C;
    cblas_dgemm(CblasColMajor, CblasNoTrans, CblasTrans,
                64, 64, 8,        // M, N, K
                1.0, _Ain, 4096, // alpha, A, LDA
                _Bin, 64,         // B, LDB
                1.0, _Cin, 512); // beta, C, LDC
  }
}
```

Listing 1: Possible implementation of $\widehat{C}_{abgcd} := \widehat{C}_{abgcd} + \widehat{A}_{cdfeg}\widehat{B}_{abegf}$ with Loop-over-GEMM. All indices have size 8.

We showed a single possible LoG implementation. However, a total of 8 implementation variants exist for (6.38): The set of possible "m"-indices is $\{(ab), a\}$, the set of "n"-indices is $\{(cd), c\}$, and the set of "k"-indices is $\{f, e\}$. Note that b is not a possible "m"-index and d is not a possible "n"-index. The reason is that the GEMM interface requires the first dimension of a matrix to have unit stride. The latter is a major limitation in the applicability of LoG, as we must not loop over the first index of $\widehat{A}, \widehat{B}$, or $\widehat{C}$.

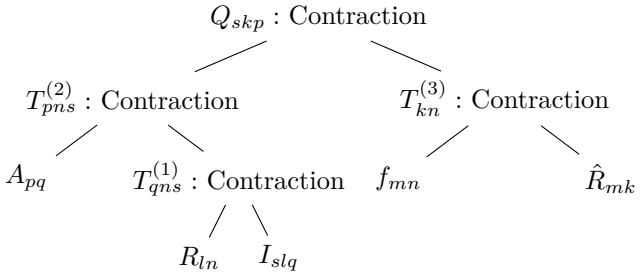


Figure 12: The figure shows an AST of $Q_{skp} := \hat{R}_{km} f_{mn} R_{ln} I_{slq} A_{pq}$. The index order of the temporary tensors $T^{(1)}$ to $T^{(3)}$ may be chosen freely by the compiler. Figure from [157].

An algorithm which lists all possible LoG-implementations is given by Springer et al. [144, Listing 7]. In YATeTo the latter algorithm is slightly extended to deal with common indices (e.g. g in (6.38)). Moreover, non-unit stride GEMMs are allowed in the case that a LoG-implementation with unit stride GEMMs is impossible. The full algorithm can be found in the software repository.[2]

6.2.5 Optimal index permutations

The Loop-over-GEMM approach is quite sensitive to index ordering. Depending on the latter, indices might be fused or might not be fused, transposes are required, or non-unit stride GEMMs are required in the worst case. In a sequence of binary tensor contractions, we need to store intermediate results in temporary tensors (see Figure 12). The index order of the temporary tensors can be chosen freely by the compiler, therefore the index order should be chosen to minimise the overall execution time.

Minimising the overall execution requires a compiler to be able to predict the overall execution time. Accurately predicting the overall execution time is difficult, even if only a single BLAS kernel is involved [120, 121]. We assume that one may come up with a sensible cost model for every node in an AST but that predicting the overall execution time of an AST is infeasible or is associated with too much effort. We therefore

[2]File yateto/ast/log.py, commit aac58baffb3431313be7490718714d420efb4780.

assume that the overall execution time of an AST can be modelled in the following way

$$\text{time} = \text{time of root node} + \text{time of children} \tag{6.40}$$

In the remainder of this section, we derive a dynamic programming problem which finds optimal index permutation for any execution time model structured as (6.40).

Dynamic programming algorithm

We formalise (6.40): Let an AST with root r be given. The set of all vertices in the tree rooted at r is given by $\mathcal{V}(r)$. Direct children of a root r are denoted with $\mathcal{C}(r)$ and the descendants of r are denoted with $\mathcal{D}(r) = \mathcal{V}(r) \setminus \{r\}$. For each vertex $v \in \mathcal{V}(r)$ a set of permissible index strings $\mathcal{P}_v$ is given. The set $\mathcal{P}_v$ is usually constructed by listing all index permutations of a node's index string, but restrictions for a node v may be enforced in this set.

For every vertex $v \in \mathcal{V}(r)$ we define a variable $x_v \in \mathcal{P}_v$, which stores the index permutation of vertex v. With the preceding definitions, the cost function (6.40) is formalised in the following recursive formula:

$$\mathrm{W}(x_r, (x_d)_{d\in\mathcal{D}(r)}) = \mathrm{w}_r\left(x_r, (x_c)_{c\in\mathcal{C}(r)}\right) + \sum_{c\in\mathcal{C}(r)} \mathrm{W}\left(x_c, (x_d)_{d\in\mathcal{D}(c)}\right). \tag{6.41}$$

The function w_r can be thought of as the cost of node r and the function W accumulates the cost of an AST.

Optimal index permutations are obtained by solving the following optimisation problem:

$$c^* = \min_{x_r\in\mathcal{P}_r, (x_d\in\mathcal{P}_d)_{d\in\mathcal{D}(r)}} \mathrm{W}(x_r, (x_d)_{d\in\mathcal{D}(r)}). \tag{6.42}$$

The optimisation problem is split in two stages. First, one minimises over the grand descendants of the root, that is, the variables x_g with

$g \in \mathcal{G}(r) = \bigcup_{c \in \mathcal{C}(r)} \mathcal{D}(c)$. Second, one minimises over the root variable and its direct children. One can show that

$$c^* = \min_{x_r, (x_c)_{c \in \mathcal{C}(r)}} \min_{(x_d)_{d \in \mathcal{G}(r)}} \mathrm{W}\left(x_r, (x_d)_{d \in \mathcal{D}(r)}\right)$$
$$= \min_{x_r, (x_c)_{c \in \mathcal{C}(r)}} \left(\mathrm{w}_r\left(x_r, (x_c)_{c \in \mathcal{C}(r)}\right) + \sum_{c \in \mathcal{C}(r)} f_c(x_c) \right), \quad (6.43)$$

where f_c is the solution to the sub-problem of finding the optimal index permutations of the AST rooted at c:

$$f_c(y) = \min_{(x_d)_{d \in \mathcal{D}(c)}} \mathrm{W}\left(y, (x_d)_{d \in \mathcal{D}(c)}\right). \quad (6.44)$$

Due to the structure of the cost function, the optimisation problem has optimal substructure, because we can find the optimal solution by combining the optimal solutions of sub-problems [28].

The solution c^* is found with a bottom up dynamic programming algorithm: In a post-order traversal of the AST, we solve (6.44) for every vertex v and for every permissible index string in $\mathcal{P}_v$. The cost of every sub-problem is memoized. In order to solve the sub-problem for v, we enumerate all possible configurations in $\mathcal{P}_v \times (\times_{c \in \mathcal{C}(v)} \mathcal{P}_c)$ and look-up the memoized cost for the sub-problems of v. The cost of the dynamic programming algorithm is in $\mathcal{O}(N(n!)^{1+c})$, where $N = |\mathcal{V}(r)|$, n is the maximum length of an index string, and c is the maximum number of children.

Cost function

The cost function w_r should ideally predict the execution time of a vertex, e.g. by employing general performance modelling techniques [120]. However, in this work a simpler approach is chosen: The cost function does only need to correctly rank several choices. That is, if one has to choose between two possible LoG-implementations, the rule of thumb that large GEMMs are more efficient than small GEMMs does already qualify as a cost function, because this rule is a proxy for execution time.

There might be other vertices than LoG (recall the multiplication and summation formulae from Section 6.2.2), but we assume that LoG dominates the overall execution time. Moreover, we assume that the following statements are true [157]:

— One should prefer unit stride GEMMs to non-unit stride GEMMs.

— One should avoid transposes of A (B) in the GEMM AB when using column-major (row-major) layout. Transposes of B (A) should be avoided if support in code generation back-ends is missing.

— Large GEMMs are more efficient than small GEMMs.

The cost function incorporates the four assumptions in the following way: The most important criterion is to avoid non-unit stride GEMMs. Then, the number of transposes shall be minimised, with preference for right-transposes. The least important criterion is to maximise the number of fused indices (fused indices lead to larger GEMMs, which is good, but it is assumed that transposes are worse than small GEMMs).

Recall that several implementations might exist for a LoG node. The cost function w_r is also used to select the best candidate from the list of possible LoG implementations (see Section 6.2.4).

Discussion

The dynamic programming algorithm is quite efficient, as we may reduce the optimisation problem to the solution of sub-problems. Still, the $n!$ in the complexity estimate is a problem in theory, as the algorithm might take a long time for high dimensional tensors. However, it is not expected that tensor dimensions in DG methods are high enough to render the dynamic programming algorithm infeasible in practice.

The cost function is a heuristic and might not always make sense. However, the dynamic programming algorithm is formulated for a very general class of cost functions and should be ready for more sophisticated cost functions, which could be implemented in future work.

6.2.6 Prefetching

Previous work on SeisSol has shown that software prefetching may improve the performance on Intel's Knights Landing architecture [64, 158]. For prefetching to be effective, `vprefetch1` instructions need to be carefully placed. The LIBXSMM library [66] offers several modes which insert prefetching instructions at suitable places during a small GEMM.

YATeTo has simple support for software prefetching: Every node in an AST has a prefetch capability, which is equal to the number of bytes which may be reached with `vprefetch1` instructions (assuming that a suitable back-end like LIBXSMM is used). The prefetch capability is computed by the `Find Prefetch Capababilities` visitor. Users may attach a list of tensors to a kernel, which shall be prefetched during the execution

of the kernel. The `Assign Prefetch` visitor then greedily assigns tensors from the prefetch list to nodes, such that the best match between prefetch capability and to-be-prefetched bytes is achieved.

6.3 Code generation

After the AST is shaped, the representation is converted to a control flow graph (CFG). The name is a euphemism for a sequence of actions without branches, but conforms with standard compiler terminology [136]. The major goal of the CFG stage is to correctly handle intermediate results and to reduce the amount of temporary buffers.

The CFG is the input to the code generation stage, where actions in the CFG are mapped to either specialised code generators, BLAS, or generic C++-code.

6.3.1 Control flow graph

Consider the following example for matrix multiplication, expressed in our DSL:

```
C['ij'] <= C['ij'] + 0.5 * A['ik'] * B['kj']
```

Clearly, this operation can be mapped to a single call to BLAS. The situation is different, though, if we replace B by C on the right-hand side. Then, we need to first multiply A and C, store the result in a temporary matrix, and finally add the temporary matrix to C.

Expressions may get much more complex than the matrix multiplication example. We thus need an algorithm which constructs a correct sequence of actions but also reduces the need for temporary tensors and useless copying of data. To this end, YATeTo adopts the model of a control flow graph [136]. The CFG consists of program points, in which the current state of the kernel is saved, and actions which connect program points and modify the state of the kernel.

The following actions are available in YATeTo:

$$X = e, \quad X = \alpha e, \quad X \mathrel{+}= e, \quad X \mathrel{+}= \alpha e. \tag{6.45}$$

The left-hand side X must be a variable, α must be a scalar, and e must be either a variable or an operation of the form $e = f(X, Y, \dots)$. Operations are taken over from the AST, e.g. Loop-over-GEMM or a multiplication formula, and they take a list of variables as argument. The equality sign serves as assignment operator, and the plus-equals stands for adding the

```
_tmp0 = LoopOverGEMM(A, B)
_tmp1 = 0.5 * _tmp0
_tmp2 = C
_tmp2 += _tmp1
C = _tmp2
```

(a) Initial CFG.

```
_tmp1 = 0.5 * LoopOverGEMM(A, B)
_tmp2 = C
_tmp2 += _tmp1
C = _tmp2
```

(b) After MergeScalarMultiplications.

```
_tmp1 = 0.5 * LoopOverGEMM(A, B)
C = C
C += _tmp1
C = C
```

(c) After SubstituteForward.

```
_tmp1 = 0.5 * LoopOverGEMM(A, B)
C += _tmp1
```

(d) After RemoveEmptyStatements.

```
C += 0.5 * LoopOverGEMM(A, B)
```

(e) After MergeActions.

Figure 13: CFG Transformations for the matrix multiplication example. Figure from [157].

left-hand side to the right-hand side and assigning the result to the left-hand side.

The initial CFG is obtained by a post-order traversal of the AST, generating a temporary variable for every node. A sequence of transformations is then employed to merge actions into compound statements, substitute variables, remove empty statements, and to reuse temporary variables. Some of these transformations are shown by way of the matrix multiplication example in Figure 13.

6.3.2 Back-end mapping

An action in the CFG is mapped to one of the following four types based on its right-hand side: `indexsum`, `product`, `copyscaleadd`, and `log`. The first two types generate code for the multiplication and summation formulae. The third type handles "simple" right-hand sides, such as scalar multiplication or transposition. The last type is responsible for Loop-over-GEMM.

A factory method exists for every type as well as a generic C++ implementation. When code is generated for an action, a type-specific description is created and is passed to the factory method. The factory method

Table 3: Feature comparison of several back-ends for GEMM.

Feature	LIBXSMM [66]	PSpaMM [14, 160]	BLAS
α	{1}	any	any
β	{0,1}	any	any
Instruction sets	SSE3, AVX, AVX2, AVX512	AVX512, NEON	many
Transpose A	no	no	yes
Transpose B	yes (since v1.11)	no	yes
sparse × dense	yes	no	no
dense × sparse	yes	yes	no

chooses a generator according to the description. For example, the generator may be chosen based on tensor size or based on the target hardware architecture. Having a factory method for every type allows to rapidly include additional code generators or to write an interface to a library.

The generic implementation of Loop-over-GEMM generates C++ loop code. For the actual calls to GEMM, the `gemm` factory method is called internally. Several generators are available for GEMM, including LIBXSMM [66], PSpaMM [14, 160], and several flavours of BLAS (MKL, OpenBLAS, BLIS). LIBXSMM and PSpaMM only support a subset of the BLAS interface but are optimised for small matrix matrix multiplications. Moreover, they offer GEMMs for sparse matrices, where the sparsity pattern is unrolled in code [14, 19]. A summary of features is shown in Table 3.

6.4 Application interface

A kernel consists of one or more assignment statements, according to the rules set up in Section 6.1.3. During compilation YATeTo generates a class for every kernel. Member variables of a kernel class are pointers to the input and output tensors involved in the kernel. Calling the kernel from C++ requires the creation of a kernel object, set the member variables to point to the respective tensors, and then call the execute member function.

The tensor toolbox supports families of kernels, too, i.e. the kernel may depend on run-time parameters. Kernel families are for example useful for the computation of surface integrals, as here the kernel depends on the geometric configuration of two adjacent tetrahedra and the kernel has

to be chosen at run-time. An example of a generated kernel interface including kernel families is shown in Listing 2.

Tightly integrated with a kernel class are flop counters. Knowing the number of flops of a kernel is useful for performance assessment. Flops are distinguished between hardware flops and non-zero flops. The hardware flop counter stores the number of flops required by the implementation on the target hardware. The non-zero flop counter stores the optimal number of flops based on equivalent sparsity patterns, that is, any multiplication with zero is excluded. The hardware flops are typically higher than the non-zero flops, because the perfect exploitation of equivalent sparsity patterns is not necessarily beneficial. In particular on architectures with wide SIMD units, exploiting sparsity does not reduce time-to-solution if one cannot fill up a vector operation [16, 64]. Nevertheless, the gap between hardware and non-zero flops give an upper bound for the speed-up one could obtain by better exploiting sparsity.

The memory layout of a tensor usually depends on the target hardware, because they are zero-padded for a particular vector width or because a sparse memory layout is useful on one architecture but not on the other. YATeTo comes with a small header-only support library which provides tensor view classes. In the application code, the user should use tensor view classes for setting tensor entries. Doing so allows to change the memory layout without having to change the application code. Furthermore, constant tensors (such as stiffness matrices of the reference element) may be included as static arrays in the generated code, such that no initialisation is required by the application code.

In addition, the compiler generates a unit test for every kernel, which checks whether the generated kernel matches the kernel's naive implementation.

6.5 Summary

In this chapter, we presented YATeTo which is designed for small tensor contractions in DG methods. Novel algorithms such as equivalent sparsity patterns and optimal index permutations are contributed, and state-of-the-art methods such as Loop-over-GEMM and code generation for small GEMMs are applied.

In Chapter 7 the integration of YATeTo in SeisSol is discussed, and the performance is evaluated in Chapter 11.

It should be emphasised that YATeTo is in no way tied to SeisSol, but may be useful for a wide range of applications. A DG spectral element

```
struct localFlux {
  constexpr static unsigned long const NonZeroFlops[] = {
    9936, 10080, 31968, 27216};
  constexpr static unsigned long const HardwareFlops[] = {
    49248, 49248, 49248, 49248};

  double const* AplusT{};
  double const* I{};
  double* Q{};
  tensor::fMrT::Container<double const*> fMrT;
  tensor::rDivM::Container<double const*> rDivM;

  struct Prefetch {
    double const* I{};
    double const* Q{};
  };
  Prefetch _prefetch;

  void execute0();
  void execute1();
  void execute2();
  void execute3();
  typedef void (localFlux::* const member_function_ptr)(void);
  constexpr static member_function_ptr ExecutePtrs[] = {
    &localFlux::execute0, &localFlux::execute1,
    &localFlux::execute2, &localFlux::execute3
  };
  constexpr static member_function_ptr findExecute(unsigned i0) {
    return ExecutePtrs[1*i0];
  }
  inline void execute(unsigned i0) {
    (this->*findExecute(i0))();
  }
  constexpr static unsigned long nonZeroFlops(unsigned i0) {
    return NonZeroFlops[1*i0];
  }
  constexpr static unsigned long hardwareFlops(unsigned i0) {
    return HardwareFlops[1*i0];
  }
};
```

Listing 2: Example kernel interface generated by YATeTo. The kernel comes in four different variants, where the parameter `i0` is used to choose a variant.

method and kernels appearing in quantum chemistry methods are discussed by Uphoff et al. [157]. These application examples are not covered in this thesis, due to the focus on earthquake simulation.

CHAPTER

7

Implementation of ADER-DG

In this chapter we take a closer look at the ADER-DG scheme and its efficient implementation.

We begin with the flux matrix decomposition in Section 7.1, which leads to a cache-aware implementation of the surface integral computation. Sections 7.2 to 7.4 demonstrate how YATeTo is used to implement ensemble simulations, viscoelasticity, and dynamic rupture. Dense and sparse memory layouts are shortly compared in Section 7.5 and we conclude with a short discussion about benefits and restrictions of YATeTo in Section 7.6.

7.1 Flux matrix decomposition

The ADER-DG scheme introduced in Chapter 3 requires flux matrices for the surface integration. The flux matrices are given by

$$F^{+}{}_{kl}^{fgh} := \int_{\mathcal{E}_2} \phi_k(\boldsymbol{\xi}^f(\boldsymbol{\chi}))\phi_l(\boldsymbol{\xi}^g(\tilde{\boldsymbol{\chi}}^h(\boldsymbol{\chi})))\,\mathrm{d}\boldsymbol{\chi}. \tag{3.24}$$

For a short repetition, f is the local face number of the tetrahedron, g is the local face number of the adjacent face in the neighbour tetrahedron, and h is the neighbour configuration. As every tetrahedron has four faces and there are three neighbour configurations, we require at most $4 \cdot 4 \cdot 3 =$

48 flux matrices. The precise number of flux matrices that are required in a simulation depends on the mesh generator. In fact, edge directions of two tetrahedra always coincide when the local vertex indices are sorted by ascending global vertex indices [132], such that only 16 flux matrices are required.

The flux matrices may take a lot of space in low level caches. Assuming these are stored densely in memory, which delivers the lowest time-to-solution on recent hardware architectures [16, 64], the flux matrices require B^2 floating point numbers, where $B := \binom{N+3}{3}$. So even in the best case of 16 matrices, degree $N = 5$ polynomials lead to a requirement of 392 KiB of cache in double precision, and 882 KiB for $N = 6$ polynomials.

The large cache requirement inflicts performance penalties, because the flux matrices get evicted from low-level caches, which forces additional memory traffic. The performance penalty is particularly large on Intel's Knights Landing architecture. As a remedy, Heinecke et al. [64] propose a prefetching scheme for the flux matrices.

The flux matrices can be decomposed analytically into a product of three smaller matrices. This flux matrix decomposition significantly reduces the cache requirement and makes prefetching of the flux matrices unnecessary on Knights Landing.

This section extends the presentation of the flux matrix decomposition from Uphoff et al. [158].

7.1.1 Analytic decomposition

We recall that ϕ_k is a polynomial with degree less or equal N, that $\boldsymbol{\xi}^f : \mathcal{E}_2 \to \mathcal{E}_3$, $\tilde{\boldsymbol{\chi}}^h : \mathcal{E}_2 \to \mathcal{E}_2$, and that $\boldsymbol{\xi}^f$ and $\tilde{\boldsymbol{\chi}}^h$ are both affine maps. Key to understanding the flux matrix decomposition is the following Lemma.

Lemma 4. *If ϕ is a polynomial on $\mathcal{E}_3$ with maximum degree less or equal N and $\boldsymbol{A} : \mathcal{E}_2 \to \mathcal{E}_3$ is an affine map, then the composition $\phi \circ \boldsymbol{A}$ is a polynomial on $\mathcal{E}_2$ with degree less or equal N.*

Proof. Every polynomial of degree less or equal N can be written as a sum of monomials, i.e.

$$\phi(\boldsymbol{\xi}) = \sum_{|\boldsymbol{\alpha}| \leq N} q_{\boldsymbol{\alpha}} \boldsymbol{\xi}^{\boldsymbol{\alpha}}.$$

where $q_{\boldsymbol{\alpha}}$ are constant coefficients and $\boldsymbol{\alpha} \in \mathbb{N}_0^3$ is a multi-index.[1] The affine map is written as $(\boldsymbol{A}(\boldsymbol{\chi}))_m = \sum_{n=1}^{2} A_{mn}\chi_n + b_m$, $m = 1, \ldots, 3$. Plugging

[1] That is, $|\boldsymbol{\alpha}| = \alpha_1 + \alpha_2 + \alpha_3$, $\boldsymbol{\xi}^{\boldsymbol{\alpha}} = \xi_1^{\alpha_1}\xi_2^{\alpha_2}\xi_3^{\alpha_3}$, and $\boldsymbol{\alpha}! = \alpha_1! \cdot \alpha_2! \cdot \alpha_3!$.

the affine map into the sum of monomials and using the multinomial theorem [1, § 24.1.2] gives

$$\phi\left(\boldsymbol{A}(\boldsymbol{\chi})\right) = \sum_{|\boldsymbol{\alpha}|\leq N} q_{\boldsymbol{\alpha}} \sum_{|\boldsymbol{\iota}|=\alpha_1} \sum_{|\boldsymbol{\kappa}|=\alpha_2} \sum_{|\boldsymbol{\lambda}|=\alpha_3} \frac{\alpha_1!}{\boldsymbol{\iota}!}\frac{\alpha_2!}{\boldsymbol{\kappa}!}\frac{\alpha_3!}{\boldsymbol{\lambda}!} \times \left(a_{11}\chi_1, a_{12}\chi_2, b_1\right)^{\boldsymbol{\iota}} \left(a_{21}\chi_1, a_{22}\chi_2, b_2\right)^{\boldsymbol{\kappa}} \left(a_{31}\chi_1, a_{32}\chi_2, b_3\right)^{\boldsymbol{\lambda}}.$$

From the above we find $\phi \circ \boldsymbol{A}$ is a linear combination of the monomials $\chi_1^{\iota_1+\kappa_1+\lambda_1}\chi_2^{\iota_2+\kappa_2+\lambda_2}$. Thus, the maximum degree is

$$|\boldsymbol{\iota}+\boldsymbol{\kappa}+\boldsymbol{\lambda}| - (\iota_3+\kappa_3+\lambda_3) \leq |\boldsymbol{\iota}+\boldsymbol{\kappa}+\boldsymbol{\lambda}| = |\boldsymbol{\alpha}| \leq N. \qquad \square$$

Using Lemma 4 we find that $\phi_k \circ \boldsymbol{\xi}^f$ is a polynomial on $\mathcal{E}_2$ with degree less or equal N. Thus, the following finite basis expansion is exact:

$$\phi_k\left(\boldsymbol{\xi}^f(\boldsymbol{\chi})\right) = R_{kl}{}^f \psi_l(\boldsymbol{\chi}), \tag{7.1}$$

where $(\psi_l)_{l=1,\dots,b}$ is a basis for polynomials with degree less or equal N on $\mathcal{E}_2$ and $b := \binom{N+2}{2}$. The coefficients $R_{kl}{}^f$ are recovered via L^2-projection:

$$R_{kl}{}^f \int_{\mathcal{E}_2} \psi_u \psi_l \,\mathrm{d}\boldsymbol{\chi} = \int_{\mathcal{E}_2} \psi_u(\boldsymbol{\chi}) \phi_k\left(\boldsymbol{\xi}^f(\boldsymbol{\chi})\right) \mathrm{d}\boldsymbol{\chi}. \tag{7.2}$$

Plugging (7.1) into (3.24) results in the flux matrix decomposition:

$$F^+_{kl}{}^{fgh} = R_{km}{}^f R_{ln}{}^g \int_{\mathcal{E}_2} \psi_u(\boldsymbol{\chi}) \psi_v\left(\tilde{\boldsymbol{\chi}}^h(\boldsymbol{\chi})\right) \mathrm{d}\boldsymbol{\chi} =: R_{ku}{}^f R_{lv}{}^g \widetilde{F}_{uv}{}^h. \tag{7.3}$$

In a similar manner, one can show that

$$F^-_{kl}{}^f = R_{ku}{}^f R_{lv}{}^f \int_{\mathcal{E}_2} \psi_u(\boldsymbol{\chi}) \psi_v(\boldsymbol{\chi}) \,\mathrm{d}\boldsymbol{\chi} =: R_{km}{}^f R_{ln}{}^f \widetilde{M}_{uv}. \tag{7.4}$$

7.1.2 Number of operations and storage requirements

Plugging the flux matrix decomposition into the discrete update scheme gives the following kernel:

$$\mathcal{I}_{lq}\mathcal{A}^-_{pq}{}^f F^+_{kl}{}^{fgh} = \mathcal{I}_{lq}\mathcal{A}^-_{pq}{}^f R_{ku}{}^f R_{lv}{}^g \widetilde{F}_{uv}{}^h. \tag{7.5}$$

It seems as one increases the work by decomposing the flux matrices, as one has to multiply five matrices instead of three at run-time. However,

Table 4: Cache consumption of the original flux matrices and the flux matrix decomposition, assuming that R and R^T are stored separately and all matrices are stored densely in double precision. The multiplication count is based on (7.6) with 9 quantities.

	Cache [KiB]			Multiplications	
N	$48 \times F^+$	$16 \times F^+$	R, f, R^T	F^+	R, f, R^T
1	6.0	2.0	1.0	468	540
2	37.5	12.5	4.6	1710	1890
3	150.0	50.0	14.8	5220	5310
4	459.4	153.1	38.1	13860	12690
5	1176.0	392.0	83.8	32760	26838
6	2646.0	882.0	165.4	70308	51660

when strength reduction is employed (see Section 6.2.2), then the optimal computation sequence with five matrices can be cheaper than the original computation sequence with 3 three matrices. Consider for example the sequence

$$R_{ku}{}^{f}\left(\left(\widetilde{F}_{uv}{}^{h}\left(R_{lv}{}^{g}\mathbb{J}_{lq}\right)\right)\mathcal{A}^{-}{}_{pq}{}^{f}\right). \tag{7.6}$$

Assuming that all matrices are dense, the number of required multiplications is readily computed to be $2bBQ + b^2Q + bQ^2$, where Q is the number of quantities. That is, we require $\mathcal{O}\left(N^5\right)$ instead of originally $\mathcal{O}\left(N^6\right)$ floating-point operations. For the elastic wave equation, where $Q = 9$, we estimate the number of multiplications in Table 4. We observe that beginning from $N = 4$ we require less multiplications, and beforehand only a small increase in multiplications is observed.

The cache requirement of the flux matrix decomposition is significantly reduced. From (7.3) we derive that we need to store the four matrices of size bB (R) and three matrices of size b^2 ($\widetilde{F}$). We assume that R is stored normal and transposed to avoid transpositions at run-time. As such, we need to store $8bB + 3b^2$ floating point numbers instead of $16B^2$ or even $48B^2$ floating point numbers (see also Table 4). Taking $N = 5$ and the Knights Landing architecture as example, we see that 16 flux matrices require 77 % of the per-core L2-cache, whereas the decomposed matrices require only 16 %.

7.1.3 Related work and conclusion

The idea to represent the boundary terms with a lower dimensional basis has also been pursued by other authors.

Atkins et al. [7] find (7.1) for a monomial basis set on a triangle. The construction of the matrix R is different, though, as it is found with the help of a symbolic algebra package [6].

Hesthaven et al. [68] use a nodal basis instead of a modal basis, i.e. a basis with the property $l_k(\boldsymbol{\xi}_l) = \delta_{kl}$. They show that if $\boldsymbol{\xi}_k$ does not reside on $\partial\mathcal{E}_3$, then $l_k(\boldsymbol{\xi}) = 0, \forall \boldsymbol{\xi} \in \partial\mathcal{E}_3$. As such, they construct an index mask, which selects boundary nodes, and then compute a flux matrix for the boundary nodes only. We note that the projection-oriented formulation in (7.2), may also be used for nodal basis functions. In particular, if $l_k(\boldsymbol{\xi}) = 0$ on the boundary, then the k-th row of the R matrices become zero. Hence, the "index mask" is automatically included.

Concluding, the projection-oriented determination of (7.1) with (7.2) is a versatile approach, because it works for all kinds of polynomial basis functions and does not require the use of symbolic algebra packages. Moreover, the discussion of cache efficiency and operation count is contributed and the original ADER-DG scheme [39] is improved for high degree polynomials, requiring less space in precious low-level caches and reducing the number of floating-point operations.

In Chapter 11, we show that the actual time-to-solution is improved and that prefetching of the flux matrices is no longer required.

7.2 Elasticity with ensemble simulations

Many recent CPU architectures offer several Single Instruction Multiple Data (SIMD) units. For example, the recent Knights Landing and Skylake server CPUs feature one or two 512-bit wide units, which allow the parallel processing of 8 double precision or 16 single precision floating point numbers.

In SeisSol, SIMD units are exploited by vectorising over the basis function coefficients. That is, the basis functions are partitioned into chunks of v, where v is the vector width, and updated in parallel. This approach works well if the number of basis functions is a multiple of the vector width or if it is large enough. However, for small orders the vectorisation strategy cannot be applied efficiently. E.g. for second order we only have 4 basis functions. So for $v = 8$ we can only utilise half of the vector width, and only a fourth for $v = 16$.

Several alternative vectorisation strategies have been investigated in the context of DG methods. E.g. Kronbichler et al. [94] vectorise over elements, and Breuer et al. [20] vectorise over simulations, i.e. they compute an ensemble of simulations within a single application run. The latter approach is particularly interesting, as the same ADER-DG on unstructured tetrahedral meshes is used as in this work. However, the approach requires an invasive change of data structure. Moreover, their vectorisation strategy requires a scenario where one has to run several simulations which are similar enough to each other, e.g. the mesh needs to identical and material parameters are shared. So an optimised code for a single simulation is still required.

In this section, we present the implementation of ensemble simulations using YATeTo. It shall be demonstrated that kernels for both the single and the ensemble simulation case can be generated with low effort.

7.2.1 Numerical scheme

Fusing S simulations into a single ensemble run requires to store the degrees of freedom for S simulations. Breuer et al. [20] propose to store the degrees of freedom in a 3D tensor, say Q_{skp}. The index s is newly introduced to index the simulation number. Setting the number of simulations S to a multiple of the vector width allows for efficient code as vectorisation over the index s always yields a full vector instruction. The numerical scheme does not change besides the newly introduced index, such that we only need to replace the degrees of freedom matrix with a 3D tensor in (3.35):

$$\begin{aligned}|J|\left({Q_{slp}}^{n+1}-{Q_{slp}}^{n}\right)M_{kl}+\sum_{f=0}^{3}\mathcal{I}_{slq}(t_{n+1},t_n)\,{\mathcal{A}^{+}_{pq}}^{f}|S_f|{F^{-}_{kl}}^{f}\\ +\sum_{f=0}^{3}{}^{m_f}\mathcal{I}_{slq}(t_{n+1},t_n)\,{\mathcal{A}^{-}_{pq}}^{f}|S_f|{F^{+}_{kl}}^{fg_fh_f}\\ -|J|\mathcal{I}_{slq}(t_{n+1},t_n)\Theta^{-1}_{ed}{A_{pq}}^{d}{K_{kl}}^{e}={\mathcal{S}_{skp}}^{n}.\end{aligned}\tag{7.7}$$

The right-hand side is slightly adjusted in comparison to (3.35): The source matrix E is zero, as we consider an elastic rheological model, and point source terms are added instead. An index s is also introduced in the point sources tensor in comparison to (3.40), which reflects that we use a different set of point sources for each simulation in the ensemble.

Note that the computational overhead of point sources can be neglected, as those only need to be added to a few individual elements.

Equation (7.7) can be further adjusted for implementation: We divide by $|J|$ and left-multiply the inverse mass matrix. Constant factors are merged in element-local matrices, the transformation of the gradient is precomputed, the flux matrix decomposition from (7.1) is inserted, and multiplication with the mass matrix is precomputed. We move all terms except ${Q_{slp}}^{n+1}$ to the right-hand side and obtain

$$\begin{aligned}{Q_{sjp}}^{n+1} = {Q_{sjp}}^{n} &+ \sum_{f=0}^{3} \mathcal{I}_{slq}(t_{n+1}, t_n)\, {\hat{\mathcal{A}}^{+}_{pq}}^{f}\, {\hat{R}_{ju}}^{f}\, {\widetilde{R}_{ul}}^{f} \\ &+ \sum_{f=0}^{3} {}^{m_f}\mathcal{I}_{slq}(t_{n+1}, t_n)\, {\hat{\mathcal{A}}^{-}_{pq}}^{f}\, {\hat{R}_{ju}}^{f}\, {R_{lv}}^{g_f}\, {\widetilde{F}_{uv}}^{h_f} \\ &+ \mathcal{I}_{slq}(t_{n+1}, t_n) {A^{*}_{pq}}^{e}\, {\hat{K}_{jl}}^{e} + \frac{1}{|J|} M^{-1}_{jk}\, {\mathfrak{S}_{skp}}^{n}, \end{aligned} \tag{7.8}$$

where we define

$$\begin{aligned} {\hat{\mathcal{A}}^{+}_{pq}}^{f} &= -\tfrac{|S_f|}{|J|} {\mathcal{A}^{+}_{pq}}^{f}, & {\hat{\mathcal{A}}^{-}_{pq}}^{f} &= -\tfrac{|S_f|}{|J|} {\mathcal{A}^{-}_{pq}}^{f}, \\ {A^{*}_{pq}}^{e} &= \Theta^{-1}_{ed} {A_{pq}}^{d}, & {\hat{K}_{jl}}^{e} &= M^{-1}_{jk} {K_{kl}}^{e}, \\ {\hat{R}_{ju}}^{f} &= M^{-1}_{jk} {R_{ku}}^{f}, & {\widetilde{R}_{ul}}^{f} &= {R_{lv}}^{f} \widetilde{M}_{uv}. \end{aligned} \tag{7.9}$$

The Cauchy-Kowalevski procedure for ensemble simulations is given by

$$|J| M_{kl}\, {\mathcal{D}_{slp}}^{i} = -|J| \Theta^{-1}_{ed} {A_{pq}}^{d} {\mathcal{D}_{slq}}^{(i-1)} {K_{lk}}^{e}. \tag{7.10}$$

We also multiply (7.10) with the inverse mass matrix and divide by $|J|$. With the definition ${\widetilde{K}_{jl}}^{e} = -M^{-1}_{jk} {K_{lk}}^{e}$ we obtain

$${\mathcal{D}_{sjp}}^{i} = {A^{*}_{pq}}^{e} {\mathcal{D}_{slq}}^{(i-1)} {\widetilde{K}_{jl}}^{e}. \tag{7.11}$$

7.2.2 Implementation and optimisation

The only difference between a single simulation and an ensemble simulation is the introduction of the index s. In Einstein notation, one prepends index s to the tensors $Q, \mathcal{I}$, and $\mathcal{D}$. Likewise we replace tensor $Q[\alpha]$ with $Q[\mathtt{s}\alpha]$ in the DSL, and proceed in the same way for tensors $\mathcal{I}$ and $\mathcal{D}$. The latter step can be automatised: Having an embedded DSL, we can override

the default behaviour of the `Tensor` class, as shown in Listing 3: The use of `OptionalDimTensor` allows, e.g., that the kernel `K['kl']*Q['lq']*A['pq']` may expand to both $K[\texttt{kl}] * Q[\texttt{lq}] * A[\texttt{pq}]$ and $K[\texttt{kl}] * Q[\texttt{slq}] * A[\texttt{pq}]$, depending on a parameter. Consequently, we only need to write the kernels once.

```
class OptionalDimTensor(Tensor):
  # ...

  def insertOptDim(self, sliceable, item):
    if self.hasOptDim():
      return sliceable[0:self._optPos] + item + \
             sliceable[self._optPos:]
    return sliceable

  def __getitem__(self, indexNames):
    indexNames = self.insertOptDim(indexNames, self._optName)
    return IndexedTensor(self, indexNames)
```

Listing 3: An embedded DSL allows to alter the behaviour of the basis building blocks. Here, an additional index is inserted automatically, such that kernels may be written only once for single and ensemble simulations.

In the remainder of this section code snippets of the implementation are shown and the most important automatic optimisations by YATeTo are discussed.

Cauchy-Kowalevski procedure: Equivalent sparsity patterns

The Cauchy-Kowalevski procedure computes time derivatives from spatial derivatives. The zeroth time derivative is represented with a polynomial of degree N. If source terms are absent, the polynomial degree decreases by one with each time derivative. So in theory the i-th time derivative can be represented exactly with $\binom{N-i+3}{3}$ basis functions. When hierarchical basis functions are employed and ordered by degree, then the stiffness tensor $\widetilde{K}_{jl}{}^{e}$ has large zero blocks. If exploited properly, these zero blocks lead to an optimal representation of the discrete time derivatives, i.e. only $\binom{N-i+3}{3}$ non-zero coefficients per quantity need to be computed [22]. The process is illustrated in Figure 14.

The implementation of the Cauchy-Kowalevski procedure is shown slightly simplified in Listing 4. The right-hand side of the kernel is built in

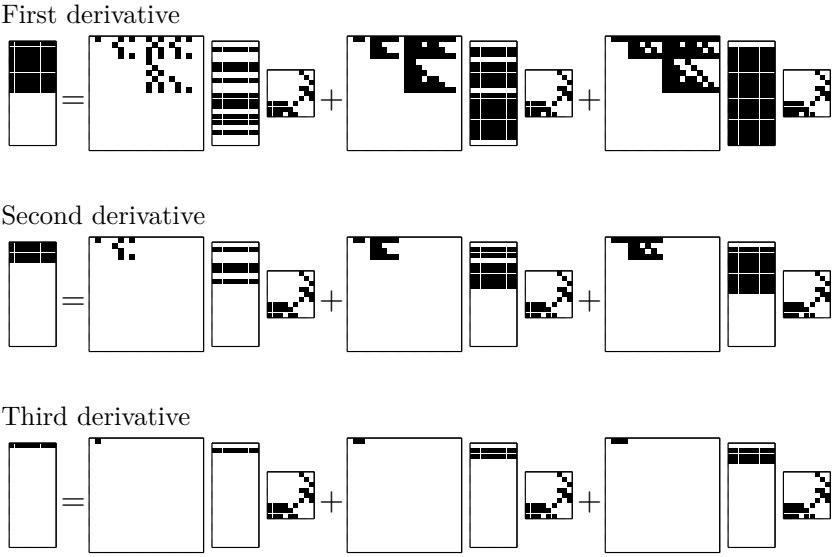


Figure 14: Equivalent sparsity patterns for the discrete Cauchy-Kowalevski procedure for a single simulation and third degree polynomials (${\mathcal{D}_{jp}}^{i} = \sum_e {\widetilde{K}_{jl}}^{e} {\mathcal{D}_{lq}}^{(i-1)} {A^*_{pq}}^{e}$). The sum over index e is unrolled, i.e. slices of the tensors K and A^* are shown.

lines 3–5, where the `Add()` command creates a child-less addition node and the `+=` operator adds child `Einsum` nodes. Note that the sum over e is written explicitly and the stiffness tensor and star tensor are given explicitly as matrices. This formulation is chosen for historical reasons. Nevertheless, the language would also allow `Ktilde['jle'] * D[-1]['lq'] * Astar['pqe']` instead, with `Ktilde` and `Astar` defined appropriately. An issue is here that YATeTo does not have a sparse format for 3D tensors yet, and so dense storage would be mandatory for $\widetilde{K}$ and A^*.

In lines 6–7 the right-hand side is evaluated and the sparsity pattern of the left-hand side is obtained. A new tensor is then defined in line 8 using the computed sparsity pattern. The latter step is crucial for the automatic deduction of vanishing coefficients in subsequent time derivative kernels. Finally, the kernel is added to the compiler in line 9.

The equivalent sparsity patterns in Figure 14 are a result of the implementation in Listing 4. In the back-end mapping step, sub-blocks are detected from the bounding boxes of the equivalent sparsity patterns,

```
D = [Q]
for i in range(1,order):
  rhs = Add()
  for e in range(3):
    rhs += Ktilde[e]['jl'] * D[-1]['lq'] * Astar[e]['pq']
  rhs = DeduceIndices( D[-1]['jp'].indices ).visit(rhs)
  rhs = EquivalentSparsityPattern().visit(rhs)
  dQ = OptionalDimTensor('dQ({})'.format(i), ..., spp=rhs.eqspp())
  generator.add('derivative({})'.format(i), dQ['jp'] <= rhs)
  D.append(dQ)
```

Listing 4: Kernel generation for the Cauchy-Kowalevski procedure.

and GEMMs operating on sub-blocks only are generated. Therefore, the vanishing derivatives optimisation is found automatically. For ensemble simulations, Loop-over-GEMM operates on sub-tensors.

Surface integrals: Strength reduction

The surface integral over the neighbouring element, i.e. the second line in (7.8), can be implemented as following:

```
kernel = lambda h,g,f: Q['jp'] <= Q['jp'] + \
  Rhat[f]['ju'] * Ftilde[h]['uv'] * R[g]['lv'] * I['lq'] * Am['pq']
prefetch = lambda h,g,f: I
generator.addFamily('neighboringFlux', \
  simpleParameterSpace(3,4,4), kernel, prefetch)
```

In lines 1–2 the kernel is defined. Tensors Q and I are of type `Optional Dim Tensor`, see Listing 3, such that the kernel definition is used for single and ensemble simulations. The kernel is here an anonymous function which depends on the parameters h, g, and f. These parameters are used to select the correct matrices analogously to (7.8). In lines 4–5 a kernel family is added to the compiler, which generates a kernel for every viable combination of parameters h, g, and f. The function call `simpleParameterSpace(3,4,4)` defines a viable combination as $(h, g, f) \in [0\,..\,2] \times [0\,..\,3] \times [0\,..\,3]$. Note that the tensor `Am` does not depend on f in contrast to (7.8), because the sparsity pattern of $\hat{A}^-$ is invariant w.r.t. f. Lastly, the family is annotated with a prefetch function defined in line 3, such that prefetch instructions for a tensor of the same shape as tensor $\mathcal{I}$ are generated. These are used to prefetch ${}^{m_{f+1}}\mathcal{I}$.

Table 5: The table shows which matrices are transposed at compile time for single and ensemble simulations.

Mode	$\widetilde{K}$	$\hat{K}$	R	$\widetilde{F}$	$\hat{R}$	$\hat{\mathcal{A}}$	A^*
Single	✗	✗	✓	✗	✗	✓	✓
Ensemble	✓	✓	✗	✓	✓	✓	✓

The complexity can be reduced from $\mathcal{O}(N^6)$ to $\mathcal{O}(N^5)$ in the surface integral kernels as pointed out in Section 7.1.2. Strength reduction finds this optimisation automatically. Interestingly, for 8–32 ensemble simulations, an order of evaluation different to (7.6) is found:

$$\left(\hat{R}_{ju}{}^{f}\,\widetilde{F}_{uv}{}^{h}\right)\left(\left(R_{lv}{}^{g}\mathcal{I}_{slq}\right)\hat{\mathcal{A}}^{-}{}_{pq}\right) \tag{7.12}$$

Optimal index permutations

We continue with surface integrals. The compiler produces the following sequence of Loop-over-GEMMs for single and ensemble simulations:

Single	Ensemble
$\alpha_{vq} := R^T_{lv}\mathcal{I}_{lq}$	$\alpha_{sv[q]} := \mathcal{I}_{sl[q]}R_{lv}$
$\beta_{uq} := \widetilde{F}_{uv}\,\alpha_{vq}$	$\beta_{(sv)p} := \alpha_{(sv)q}(\hat{\mathcal{A}}^{-})^T_{pq}$
$\gamma_{up} := \beta_{uq}(\hat{\mathcal{A}}^{-})^T_{pq}$	$\gamma_{jv} := \hat{R}_{ju}\widetilde{F}_{uv}$
$Q_{jp} := Q_{jp} + \hat{R}_{ju}\gamma_{up}$	$Q_{sj[p]} := Q_{sj[p]} + \beta_{sv[p]}\gamma^T_{jv}$

Greek letters denote temporary tensors here and the parameters h, g, f are dropped for clarity. The optimal index permutation algorithm from Section 6.2.5 minimises the heuristic cost function: Non-unit stride GEMMs are absent, transposes are minimised, and indices are fused.

Single and ensemble simulations differ: In the former variant we multiply the transpose of R from the left whereas in the latter variant we multiply R from the right without transposition. The matrix R is constant and as such we may store a transposed copy of R already at compile time. As such, transposes at run-time can be avoided.

Tensors provided by the user are taken "as is". YATeTo only finds good index permutations for intermediate results. Nevertheless, the mapping to Loop-over-GEMM can be inspected by the user and as such the toolbox

Table 6: Relative overhead of two implementation variants for viscoelasticity compared to elasticity on Intel Haswell CPUs [156].

Mechanisms	0	1	3	5	7	9
Quantities	9	15	27	39	51	63
Matrix chain	1.00	1.47	1.96	2.46	2.98	3.72
Kronecker	1.00	1.48	1.75	2.06	2.41	2.66

provides guidance on the index permutations of tensors. In SeisSol, the permutation analysis lead to the implementation decisions summarised in Table 5.

7.3 Viscoelasticity

Considering the number of PDEs, the introduction of viscoelastic attenuation comes at a high price. In addition to the 9 PDEs required in the elastic case, an additional $6L$ PDEs, L being the number of mechanisms, are required, as detailed in Section 2.2. But Equations (2.28) to (2.30) exhibit structure: Spatial derivatives w.r.t. memory variables are not required and the 6 equations which make up a mechanisms are identical up to the relaxation frequency.

The lack of spatial derivatives w.r.t. memory variables is reflected by large zero blocks in the coefficient matrices ${A_{pq}}^{d}$. Uphoff et al. [156] show that exploiting zero blocks, using equivalent sparsity patterns, leads to an implementation whose time-to-solution grows less than linear with number of quantities. Moreover, they present a second implementation variant in which the coefficient matrices are written in terms of Kronecker products leading to another speed-up of up to 1.4. The relative overheads of the two variants are summarised in Table 6.

In both implementation variants, the degrees of freedom for the memory variables are stored in a matrix along with the degrees of freedom for the elastic quantities. The memory variables, however, do not take part in the solution of the Riemann problem at the interfaces (see Section 4.2.2), and as such they are only required in element-local operations. It is not necessary to share memory variables with neighbouring ranks in a distributed memory system. Furthermore, the previous code generator [156] only supports matrix chain products, such that the Kro-

necker implementation required some operations to be manually mapped to linear algebra operations. In YATeTo support for higher dimensional tensors is built-in. Therefore a third implementation variant is proposed in which the degrees of freedom are split into a quantity matrix and a 3D memory variable tensor. This implementation avoids unnecessary data transfers in a distributed memory setting and all tensor operations are automatically mapped to linear algebra operations by the compiler. In order to distinguish the three implementation variants we call them the matrix chain implementation, the unsplit Kronecker implementation, and the split Kronecker implementation, respectively.

We first review the matrix chain implementation in Section 7.3.1. The split Kronecker implementation is presented in Section 7.3.2. Details of the unsplit Kronecker implementation [156] are omitted.

7.3.1 Matrix chain implementation

The viscoelastic wave equation can be brought in the standard form (3.1). Following Käser et al. [81], the coefficient matrices are

$$\breve{A}_{::}{}^{d} = \begin{pmatrix} A_{::}{}^{d} & 0 \\ \breve{B}_{::}{}^{d} & 0 \end{pmatrix} \in \mathbb{R}^{n_v \times n_v}, \tag{7.13}$$

where $n_v = 9 + 6L$. We use the colon to refer to a fibre and two colons to refer to a slice of a tensor. The tensor A is defined identical to the elastic wave equation and the tensor $\breve{B} \in \mathbb{R}^{6L \times 9 \times 3}$ is given by

$$\breve{B}_{::}{}^{d} = \omega \otimes B_{::}{}^{d}. \tag{7.14}$$

The column vector $\omega \in \mathbb{R}^{L}$ contains the relaxation frequencies and the non-zeros of $B \in \mathbb{R}^{6 \times 9 \times 3}$ are given by

$$\begin{aligned} B_{17}{}^{1} &= -1 & B_{48}{}^{1} &= -\tfrac{1}{2} & B_{69}{}^{1} &= -\tfrac{1}{2} \\ B_{28}{}^{2} &= -1 & B_{47}{}^{2} &= -\tfrac{1}{2} & B_{59}{}^{2} &= -\tfrac{1}{2} \\ B_{39}{}^{3} &= -1 & B_{58}{}^{3} &= -\tfrac{1}{2} & B_{67}{}^{3} &= -\tfrac{1}{2} \end{aligned} \tag{7.15}$$

A source matrix is required to account for the right-hand side of the viscoelastic wave equation (Equations (2.28) and (2.30)). The source matrix is given by

$$\breve{E}_{::}{}^{d} = \begin{pmatrix} 0 & E_{::1} & \dots & E_{::L} \\ 0 & -\omega_1 I & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & -\omega_L I \end{pmatrix} \in \mathbb{R}^{n_v \times n_v} \tag{7.16}$$

$I \in \mathbb{R}^{6\times 6}$ is an identity matrix and the tensor $E \in \mathbb{R}^{9\times 6\times L}$ is defined as

$$E_{::l} = \begin{pmatrix} -Y^{\lambda}_{l} - 2Y^{\mu}_{l} & -Y^{\lambda}_{l} & -Y^{\lambda}_{l} & 0 & 0 & 0 \\ -Y^{\lambda}_{l} & -Y^{\lambda}_{l} - 2Y^{\mu}_{l} & -Y^{\lambda}_{l} & 0 & 0 & 0 \\ -Y^{\lambda}_{l} & -Y^{\lambda}_{l} & -Y^{\lambda}_{l} - 2Y^{\mu}_{l} & 0 & 0 & 0 \\ 0 & 0 & 0 & -2Y^{\mu}_{l} & 0 & 0 \\ 0 & 0 & 0 & 0 & -2Y^{\mu}_{l} & 0 \\ 0 & 0 & 0 & 0 & 0 & -2Y^{\mu}_{l} \end{pmatrix}. \tag{7.17}$$

The last missing piece is the numerical flux. We use the rotational invariance property of the viscoelastic wave equation with the transformation matrix $\breve{T}$, which has the following structure [81]:

$$\breve{T}(\boldsymbol{n}) := \begin{pmatrix} T^t(\boldsymbol{n}) & & & & \\ & T^v(\boldsymbol{n}) & & & \\ & & T^t(\boldsymbol{n}) & & \\ & & & \ddots & \\ & & & & T^t(\boldsymbol{n}) \end{pmatrix} \in \mathbb{R}^{n_v \times n_v}, \tag{7.18}$$

where $T^t \in \mathbb{R}^{6\times 6}$ transforms tensors and $T^v \in \mathbb{R}^{3\times 3}$ transforms vectors from a basis for an interface with normal $\boldsymbol{n}$ to the physical basis. Using the results from Section 4.2.2 we obtain the following flux solvers:

$$\begin{aligned} \breve{\mathcal{A}}^{+}_{pq}{}^{f} &:= \breve{T}_{pa}(\boldsymbol{n}_f)\breve{A}_{ab}{}^{1}(\breve{R}^{f})_{bc}\breve{\chi}^{+}_{cd}(\breve{R}^{f})^{-1}_{de}\breve{T}^{-1}_{eq}(\boldsymbol{n}_f) \\ \breve{\mathcal{A}}^{-}_{pq}{}^{f} &:= \breve{T}_{pa}(\boldsymbol{n}_f)\breve{A}_{ab}{}^{1}(\breve{R}^{f})_{bc}\breve{\chi}^{-}_{cd}(\breve{R}^{f})^{-1}_{de}\breve{T}^{-1}_{eq}(\boldsymbol{n}_f) \end{aligned} \tag{7.19}$$

Note that $\breve{R}^{f}$ is called $\breve{R}^{\pm}$ in Section 4.2.2 and depends on the material parameters of element m and the adjacent element m_f at the f-th face.

As evident from (4.33) and (7.19), the flux solvers have the following structure:

$$\breve{\mathcal{A}}^{\pm}{}_{::}{}^{f} = \begin{pmatrix} \mathcal{A}^{\pm}{}_{::}{}^{f} & 0 \\ \omega \otimes \mathcal{B}^{\pm}{}_{::}{}^{f} & 0 \end{pmatrix} \in \mathbb{R}^{n_v \times n_v}, \tag{7.20}$$

where $\mathcal{A}^{\pm}{}_{::}{}^{f} \in \mathbb{R}^{9\times 9}$ and $\mathcal{B}^{\pm}{}_{::}{}^{f} \in \mathbb{R}^{6\times 9}$.

The implementation of the scheme is now straightforward: In Equation (7.8) we replace tensors $A_{pq}{}^{d}$ and $\mathcal{A}^{\pm}{}_{pq}{}^{f}$ by tensors $\breve{A}_{pq}{}^{d}$ and $\breve{\mathcal{A}}^{\pm}{}_{pq}{}^{f}$. In addition, we have to add the following source term to the right-hand side of Equation (7.8):

$$\breve{E}_{pq}{}^{m}\mathcal{I}_{jq}({}^{m}t_{n+1} - {}^{m}t_{n}). \tag{7.21}$$

We also need to add the source term to the Cauchy-Kowalevski procedure:

$$\mathcal{D}_{jp}{}^{i} = \breve{A}^{*}{}_{pq}{}^{e}\mathcal{D}_{lq}{}^{(i-1)}\widetilde{K}_{jl}{}^{e} + \breve{E}_{pq}\mathcal{D}_{jq}{}^{(i-1)}. \tag{7.22}$$

In contrast to Figure 14, the coefficients do not vanish in the above Cauchy-Kowalevski procedure, which is automatically reflected in the equivalent sparsity patterns.

7.3.2 Split Kronecker implementation

We begin with rewriting the viscoelastic wave equation, cf. Equations (2.28) to (2.30), in the following form:

$$\frac{\partial q_p}{\partial t} + A_{pq}{}^{d}\frac{\partial q_q}{\partial x_d} = E_{pqr}p_{qr}, \tag{7.23}$$

$$\frac{\partial p_{pr}}{\partial t} + \omega_r B_{pq}{}^{d}\frac{\partial q_q}{\partial x_d} = -\omega_r p_{pr}. \tag{7.24}$$

The vector of quantities q_p and the coefficient matrices $A_{pq}{}^{d}$ are defined as for the elastic wave equation [39]. The memory variable matrix is defined by $p_{:r} = (\zeta_{11r}, \zeta_{22r}, \zeta_{33r}, \zeta_{12r}, \zeta_{23r}, \zeta_{13r})$. Tensors B and E are defined in (7.15) and (7.17).

The memory variables are discretised with the finite basis expansion

$$p_{pr}(\boldsymbol{x}, t) = {}^{m}P_{lpr}(t)\phi_l({}^{m}\boldsymbol{\Xi}(\boldsymbol{x})), \tag{7.25}$$

i.e. in the same way as in Chapter 3 but with an additional dimension for the mechanisms.

7.3.3 Discrete update scheme

Deriving the discrete update scheme for (7.23) is very similar to the derivation of (7.8), therefore we omit the details and state the final form:

$$\begin{aligned} Q_{jp}{}^{*} &= Q_{jp}{}^{n} + \sum_{f=0}^{3} \mathcal{I}^{Q}{}_{lq} \hat{\mathcal{A}}^{+}{}_{pq}{}^{f} \hat{R}_{ju}{}^{f} \widetilde{R}_{ul}{}^{f} \\ &\quad + \mathcal{I}^{Q}{}_{lq} A^{*}{}_{pq}{}^{e} \hat{K}_{jl}{}^{e} + \mathcal{I}^{P}{}_{jqr} E_{pqr} + \frac{1}{|J|} M^{-1}_{jk} \mathcal{S}_{kp}{}^{n}, \\ Q_{jp}{}^{n+1} &= Q_{jp}{}^{*} + \sum_{f=0}^{3} {}^{m_f}\mathcal{I}^{Q}{}_{lq} \hat{\mathcal{A}}^{-}{}_{pq}{}^{f} \hat{R}_{ju}{}^{f} R_{lv}{}^{g_f} \widetilde{F}_{uv}{}^{h_f}. \end{aligned} \tag{7.26}$$

Note that the time-integrated degrees of freedom are split into an elastic part $\mathcal{I}^{Q}$ and an anelastic part $\mathcal{I}^{P}$. In addition, we distinguish between the local update step (first equation) and the neighbour update step (second equation).

The update scheme for (7.24) is

$$\begin{aligned} P_{jpr}{}^{*} &= P_{jpr}{}^{n} + \sum_{f=0}^{3} \mathcal{I}^{Q}{}_{lq} \omega_r \hat{\mathcal{B}}^{+}{}_{pq}{}^{f} \hat{R}_{ju}{}^{f} \widetilde{R}_{ul}{}^{f} \\ &\quad + \mathcal{I}^{Q}{}_{lq} \omega_r B^{*}{}_{pq}{}^{e} \hat{K}_{jl}{}^{e} - \omega_r \mathcal{I}^{P}{}_{jpr}, \\ P_{jpr}{}^{n+1} &= P_{jpr}{}^{*} + \sum_{f=0}^{3} {}^{m_f}\mathcal{I}^{Q}{}_{lq} \omega_r \hat{\mathcal{B}}^{-}{}_{pq}{}^{f} \hat{R}_{ju}{}^{f} R_{lv}{}^{g_f} \widetilde{F}_{uv}{}^{h_f}, \end{aligned} \tag{7.27}$$

where we define $B^{*}{}_{pq}{}^{e} = \Theta^{-1}_{ed} B_{pq}{}^{d}$ and $\hat{\mathcal{B}}^{\pm}{}_{pq}{}^{f} = -\frac{|S_f|}{|J|} \mathcal{B}^{\pm}{}_{pq}{}^{f}$.

Both update schemes have common terms, such as $\mathcal{I}^{Q}{}_{lq} \hat{K}_{jl}{}^{e}$, and we should compute those only once. Automatic common subexpression elimination is currently not implemented in YATeTo (which is a challenge in itself [63]). Therefore the following alternative is considered: We define the extended matrices

$$\begin{aligned} \{AB\}^{*}{}_{::}{}^{e} &:= \begin{pmatrix} A^{*}{}_{::}{}^{e} & B^{*}{}_{::}{}^{e} \end{pmatrix} \in \mathbb{R}^{9 \times 15}, \\ \{\hat{\mathcal{A}}\hat{\mathcal{B}}\}^{\pm}{}_{::}{}^{f} &:= \begin{pmatrix} \hat{\mathcal{A}}^{\pm}{}_{::}{}^{f} & \hat{\mathcal{B}}^{\pm}{}_{::}{}^{f} \end{pmatrix} \in \mathbb{R}^{9 \times 15}, \end{aligned} \tag{7.28}$$

and define the extended local update step

$$\{Q\}_{jp}{}^{*} = \sum_{f=0}^{3} \mathcal{I}^{Q}{}_{lq} \{\hat{\mathcal{A}}\hat{\mathcal{B}}\}^{+}{}_{pq}{}^{f} \hat{R}_{ju}{}^{f} \widetilde{R}_{ul}{}^{f} + \mathcal{I}^{Q}{}_{lq} \{AB\}^{*}{}_{pq}{}^{e} \hat{K}_{jl}{}^{e}, \tag{7.29}$$

and the extended neighbour update step

$$\{Q\}_{jp}{}^{n+1} = \sum_{f=0}^{3} {}^{m_f}\mathcal{I}^{Q}{}_{lq} \{\hat{\mathcal{A}}\hat{\mathcal{B}}\}^{-}{}_{pq}{}^{f} \hat{R}_{ju}{}^{f} R_{lv}{}^{g_f} \widetilde{F}_{uv}{}^{h_f}. \tag{7.30}$$

The first 9 columns of $\{Q\}^{*}$ respectively $\{Q\}^{n+1}$ contain the update for the elastic part and the last 6 columns contain the update for the anelastic part. We recover these parts with the selection matrices

$$\mathfrak{S}^{\text{ela}} = \begin{pmatrix} I_9 \\ 0_{6\times 9} \end{pmatrix}, \qquad \mathfrak{S}^{\text{ane}} = \begin{pmatrix} 0_{9\times 6} \\ I_6 \end{pmatrix}. \tag{7.31}$$

Plugging the extended tensors into the update schemes gives

$$\begin{aligned} Q_{jp}{}^{*} &= Q_{jp}{}^{n} + \{Q\}_{jq}{}^{*} \mathfrak{S}^{\text{ela}}{}_{qp} + \mathcal{I}^{P}{}_{jqr} E_{pqr} + \frac{1}{|J|} M^{-1}_{jk} \mathcal{S}_{kp}{}^{n}, \\ Q_{jp}{}^{n+1} &= Q_{jp}{}^{*} + \{Q\}_{jq}{}^{n+1} \mathfrak{S}^{\text{ela}}{}_{qp}, \\ P_{jpr}{}^{*} &= P_{jpr}{}^{n} + \omega_r \{Q\}_{jq}{}^{*} \mathfrak{S}^{\text{ane}}{}_{qp} - \omega_r \mathcal{I}^{P}{}_{jpr}, \\ P_{jpr}{}^{n+1} &= P_{jpr}{}^{*} + \omega_r \{Q\}_{jq}{}^{n+1} \mathfrak{S}^{\text{ane}}{}_{qp}. \end{aligned} \tag{7.32}$$

7.3.4 Cauchy-Kowalevski procedure

Constructing the discrete Cauchy-Kowalevski procedure for (7.23) and (7.24) gives

$$\begin{aligned} \mathcal{D}^{Q}{}_{jp}{}^{i} &= A^{*}{}_{pq}{}^{e} \mathcal{D}^{Q}{}_{lq}{}^{(i-1)} \widetilde{K}_{jl}{}^{e} + E_{pqr} \mathcal{D}^{P}{}_{jqr}{}^{(i-1)}, \\ \mathcal{D}^{P}{}_{jpr}{}^{i} &= \omega_r B^{*}{}_{pq}{}^{e} \mathcal{D}^{Q}{}_{lq}{}^{(i-1)} \widetilde{K}_{jl}{}^{e} - \omega_r \mathcal{D}^{P}{}_{jpr}{}^{(i-1)}, \end{aligned} \tag{7.33}$$

where the derivatives are split into an elastic part $\mathcal{D}^{Q}$ and an anelastic part $\mathcal{D}^{P}$.

We repeat the common subexpression elimination step from Section 7.3.3 and obtain

$$\begin{aligned} \{\mathcal{D}\}_{jp}{}^{i} &= \{AB\}^{*}{}_{pq}{}^{e}\mathcal{D}^{Q}{}_{lq}{}^{(i-1)}\widetilde{K}_{jl}{}^{e}, \\ \mathcal{D}^{Q}{}_{jp}{}^{i} &= \{\mathcal{D}\}_{jq}{}^{i}\mathfrak{S}^{\text{ela}}{}_{qp} + E_{pqr}\mathcal{D}^{P}{}_{jqr}{}^{(i-1)}, \\ \mathcal{D}^{P}{}_{jpr}{}^{i} &= \omega_r\{\mathcal{D}\}_{jq}{}^{i}\mathfrak{S}^{\text{ane}}{}_{qp} - \omega_r\mathcal{D}^{P}{}_{jpr}{}^{(i-1)}. \end{aligned} \tag{7.34}$$

7.3.5 Implementation

The implementation and optimisation steps are very similar to Section 7.2.2. We note that $E_{p(qr)}$ is transposed at compile-time to $E_{(qr)p}$.

7.4 Dynamic rupture

We introduced dynamic rupture as boundary conditions on the fault surface Σ (Section 2.4.2). In the numerical scheme, these conditions are imposed via non-linear numerical fluxes (Sections 3.3 and 4.3.1). The surface Σ is triangulated and each triangle in Σ is adjacent to the two tetrahedra. For these tetrahedra, we describe the computation of the non-linear numerical fluxes, which consists of the following three steps:

1. Compute space-time interpolation points.
2. Solve the inverse Riemann problem for each interpolation point and compute the flux.
3. Integrate the flux on the fault in space and time.

Step 1. Recall that we predict the evolution in time of the degrees of freedom with a Taylor expansion. Given derivatives $\mathcal{D}_{lp}{}^{i}$ we have

$$Q_{lp}(\tau_z) = \sum_{d=0}^{N} \frac{(\tau_z)^d}{d!}\mathcal{D}_{lp}{}^{d}, \tag{7.35}$$

where τ_i are quadrature points for the time interval $[t_n; t_{n+1}]$. For the interpolation in space, we define the generalised Vandermonde matrices

$$\begin{aligned} V^{-}{}_{il}{}^{f} &:= \phi_l(\boldsymbol{\xi}^f(\boldsymbol{\chi}_i)), \\ V^{+}{}_{il}{}^{gh} &:= \phi_l(\boldsymbol{\xi}^g(\tilde{\boldsymbol{\chi}}^h(\boldsymbol{\chi}_i))), \end{aligned} \tag{7.36}$$

where $\boldsymbol{\chi}_i$ are quadrature points on the reference triangle $\mathcal{E}_2$. The space-time interpolation is compactly written as

$$\begin{aligned} {q^-}_p{}^f(\boldsymbol{\chi}_i, \tau_z) &= {V^-}_{il}{}^f \left(\sum_{d=0}^{N} \frac{(\tau_z)^d}{d!} \; {}^m\mathcal{D}_{lp}{}^d \right), \\ {q^+}_p{}^f(\boldsymbol{\chi}_i, \tau_z) &= {V^+}_{il}{}^{g_f h_f} \left(\sum_{d=0}^{N} \frac{(\tau_z)^d}{d!} \; {}^{m_f}\mathcal{D}_{lp}{}^d \right). \end{aligned} \tag{7.37}$$

We use the superscripts plus and minus to indicate the side of the interface. These superscript are related to the definition of the normal vector, i.e. the side in negative normal direction and the side in positive normal direction. This notation is used in the seminal work of Day et al. [34] and is consistent with the remainder of this thesis. (Some authors use the opposite convention [123, 124, 158].) Moreover, note that the element numbers m and m_f and the face parameters g_f and h_f are used: The numerical flux is computed as seen from the minus-side and the face parameters are determined as in Section 3.1.3.

Step 2. We assume that the Riemann solver at the fault is given by the two functions $\boldsymbol{w}^b(\boldsymbol{w}^{-f}, \boldsymbol{w}^{+f}; \boldsymbol{\psi})$ and $\boldsymbol{w}^c(\boldsymbol{w}^{-f}, \boldsymbol{w}^{+f}; \boldsymbol{\psi})$. These functions return the interface states in a fault-aligned coordinate system. (Note that $\boldsymbol{w}^b \neq \boldsymbol{w}^c$ in general as a velocity discontinuity is modelled, and $\boldsymbol{\psi}$ contains the state variables; see Section 4.3.1 for details.) The space-time interpolation is given in the physical coordinate system and needs to be transformed to the fault-aligned coordinate system:

$$\begin{aligned} {w^-}_p{}^f(\boldsymbol{\chi}_i, \tau_z) &= T^{-1}_{pq}(\boldsymbol{n}_f) {q^-}_q{}^f(\boldsymbol{\chi}_i, \tau_z), \\ {w^+}_p{}^f(\boldsymbol{\chi}_i, \tau_z) &= T^{-1}_{pq}(\boldsymbol{n}_f) {q^+}_q{}^f(\boldsymbol{\chi}_i, \tau_z). \end{aligned} \tag{7.38}$$

The Riemann solvers are evaluated at each space-time interpolation point and saved in the following two tensors:

$$\begin{aligned} {\mathcal{W}^-}_{ipz}{}^f &:= {w^b}_p \left(\boldsymbol{w}^{-f}(\boldsymbol{\chi}_i, \tau_z), \boldsymbol{w}^{+f}(\boldsymbol{\chi}_i, \tau_z); \boldsymbol{\psi}(\boldsymbol{\chi}_i, \tau_z) \right), \\ {\mathcal{W}^+}_{ipz}{}^f &:= {w^c}_p \left(\boldsymbol{w}^{-f}(\boldsymbol{\chi}_i, \tau_z), \boldsymbol{w}^{+f}(\boldsymbol{\chi}_i, \tau_z); \boldsymbol{\psi}(\boldsymbol{\chi}_i, \tau_z) \right). \end{aligned} \tag{7.39}$$

We need to transform ${\mathcal{W}^-}_{ipz}{}^f$ and ${\mathcal{W}^+}_{ipz}{}^f$ to the physical coordinate system with matrix T_{pq} (the $\boldsymbol{n}_f$ dependency is dropped for readabil-

ity in the following). Together with the rotational invariance property, $n_d A_{pq}{}^d = T_{pr}\, A_{rs}{}^1 T^{-1}_{sq}$, the fluxes at the interpolation points are

$$T_{pr}\,{}^m A_{rq}{}^1 \mathcal{W}^-{}_{iqz}{}^f \quad \text{and} \quad T_{pr}\,{}^{m_f} A_{rq}{}^1 \mathcal{W}^+{}_{iqz}{}^f. \tag{7.40}$$

Step 3. The flux integration uses quadrature rules in space and time (see Section 3.3). On the minus side we have

$$\mathcal{F}^-{}_{kp}{}^f \approx T_{pr}\,{}^m A_{rq}{}^1 |S_f| \sum_{i=1}^{N^s} \sum_{z=1}^{N^t} \beta_i \gamma_z \phi_k(\boldsymbol{\xi}^f(\boldsymbol{\chi}_i)) \mathcal{W}^-{}_{iqz}{}^f \tag{7.41}$$

We save computations in above formula by first integrating in time. Moreover we reuse tensor V and obtain

$$\mathcal{F}^-{}_{kp}{}^f \approx T_{pr}\,{}^m A_{rq}{}^1 |S_f| \sum_{i=1}^{N^s} \beta_i V^-{}_{ik}{}^f \sum_{z=1}^{N^t} \gamma_z \mathcal{W}^-{}_{iqz}{}^f. \tag{7.42}$$

In the discrete update scheme the above needs to be multiplied with the inverse mass matrix M^{-1}_{jk}, divided by ${}^m|J|$, and moved to the right-hand side (recall Section 7.2.2). We define

$$\hat{A}^-{}_{pq}{}^f := -\frac{|S_f|}{{}^m|J|} T_{pr}\,{}^m A_{rq}{}^1, \qquad \widehat{V}^-{}_{ji}{}^f := M^{-1}_{jk} \beta_i V^-{}_{ik}{}^f, \tag{7.43}$$

and compactly state the update for the degrees of freedom on the minus side as following:

$$-\frac{1}{{}^m|J|} M^{-1}_{jk} \mathcal{F}^-{}_{kp}{}^f \approx \hat{A}^-{}_{pq}{}^f \widehat{V}^-{}_{ji}{}^f \sum_{z=1}^{N^t} \gamma_z \mathcal{W}^-{}_{iqz}{}^f. \tag{7.44}$$

We proceed in a similar manner for the derivation of the plus side update. Here, we have to keep in mind that everything so far is based on the normal pointing from the minus-side to the plus-side, which is the *inward* pointing normal for the plus side. Therefore, we have to flip the sign. The update for the plus side is given by

$$\frac{1}{{}^{m_f}|J|} M^{-1}_{jk} \mathcal{F}^+{}_{kp}{}^f \approx \hat{A}^-{}_{pq}{}^f \widehat{V}^+{}_{ji}{}^{g_f h_f} \sum_{z=1}^{N^t} \gamma_z \mathcal{W}^+{}_{iqz}{}^f, \tag{7.45}$$

where we defined

$$\hat{A}^{-}{}_{pq}{}^{f} := \frac{|S_f|}{m_f|J|} T_{pr}{}^{m_f} A_{rq}{}^{1}, \qquad \widehat{V}^{-}{}_{ji}{}^{gh} := M^{-1}_{jk} \beta_i V^{+}{}_{ik}{}^{gh}. \tag{7.46}$$

In this section, it is shown how to compute the non-linear numerical flux for two adjacent tetrahedra simultaneously. On a parallel computer one needs to be careful to avoid a race condition when adding the updates to the two tetrahedra. We come back to this issue in Section 8.2.

7.5 Memory layouts

YATeTo supports sparse matrices in the CSC format and we have to decide on a memory layout for each matrix. The latter choice may vary for each hardware architecture and polynomial degree. In previous work, memory layouts are chosen with auto-tuning [16, 156, 158].

The performance of sparse matrix multiplication is rather mediocre on recent hardware architectures with the exception of dense $\times$ sparse multiplication, as briefly discussed in Section 6.2.3. In fact, an experiment by Brei [14] suggests that a properly register-blocked dense $\times$ sparse matrix multiplication could be always faster than its dense $\times$ dense counterpart, as long as the number of instructions is not too large. Therefore, instead of auto-tuning it seems appropriate to adopt the following simpler strategy: Choose a dense memory layout in the sparse $\times$ dense case, and a CSC memory layout in the dense $\times$ sparse case.

In Chapter 11, auto-tuning is used for the Shaking Corals version [158], and the mentioned simple strategy is used for the YATeTo version. Note that auto-tuning may be combined with YATeTo, using either the existing strategies [16, 156] or a generic auto-tuning framework [5].

7.6 Einstein notation – a proper abstraction?

The development of YATeTo started with the need to handle software complexity. But is the level of abstraction chosen appropriate?

On the one hand, kernels written in the DSL closely resemble the mathematical formulation of the numerical scheme. Major optimisation steps are reproduced automatically, such as vanishing derivatives (via equivalent sparsity patterns) and the optimal order of evaluation (via strength reduction). Equivalent sparsity patterns also automatically exploit the structure of the viscoelastic wave equation (cf. Figure 10). As YATeTo's DSL

is embedded into Python, missing language features can be quickly substituted with meta-programming techniques, such as the introduction of the `OptionalDimTensor` or kernel family generation with lambda expressions. Therefore, the language itself can be kept compact. The abstraction is sufficiently low-level. Memory layout, storage order, pre-computation of terms, prefetching, and GEMM back-ends may be controlled precisely. Moreover, due to the support of a broad spectrum of tensor operations, the numerical scheme does not need to be formulated in a canonical form. Different implementation strategies, such as splitting quantities for viscoelasticity, can be tested quickly.

On the other hand, many steps in the process of transforming the weak form of the system of PDEs to code have to be done manually. E.g. the mapping to the reference element, the parameterisation of the surface integrals, or the choice of numerical flux. The pre-computation of terms needs to be done manually, and index permutations need to be chosen manually to some extent (YATeTo only finds optimal permutations for temporary tensors). While specifying all tensor indices gives control over the index permutations, having to write them out might lead to less generic code. E.g. when a tensor basis is used,[2] the only difference between a scheme in two spatial dimensions and a scheme in three spatial dimensions is an additional index, which has to be inserted in a rather mechanical manner. Here, other notation, such as n-mode products [86], might be more adequate.

In the author's opinion, YATeTo is a well-suited abstraction for tensor operations and is sufficiently expressive for low-level optimisations. However, when starting from scratch for a different system of PDEs, some time is likely spent in the formalisation of the numerical scheme as tensor contractions. An interesting research direction would thus be to automatically transform a weak form to a numerical scheme in Einstein notation, i.e. mimicking the process from transforming (3.1) to, e.g., (7.8). Previous research on how to transform a weak form into code resulted in the Unified Form Language [4], which is now widely used and has been integrated into various frameworks such as Firedrake [127] or Dune [84]. Therefore, the next logical step would be to evaluate whether YATeTo could serve as a "back-end" for one of these frameworks, or if one might integrate the optimal index permutation algorithm, the equivalent sparsity pattern algorithm, prefetching, or the map to loop-over-GEMM algorithms in an existing framework.

[2] See the LinA example discussed by Uphoff et al. [157].

CHAPTER

8

Local time-stepping for dynamic rupture

Time is advanced using an explicit method in SeisSol. In order to maintain stability, an element's time-step is limited as following [39, 41]:

$$\Delta t_{\max} < \frac{1}{2N+1} \frac{d_{\text{in}}}{\lambda_{\max}}, \tag{8.1}$$

where d_{in} is the diameter of the element's insphere, $\lambda_{\max}$ is the maximum wave-speed, and N is the polynomial degree.

A problematic issue in unstructured mesh generation are slivers [41, and references therein], which are almost planar tetrahedra with a very small insphere diameter. Dynamic rupture problems may impose strong requirements on the mesh generator. In particular, slivers might be introduced by the mesh generator when a fault intersects the free surface at a shallow angle (e.g. in a subduction zone). When global time-stepping is used, that is, the element with the smallest time-step dictates the time-step of all elements, then a single sliver is sufficient to deteriorate the efficiency or even feasibility of a scenario.

Local time-stepping (LTS) is a method to overcome such issues. A single sliver only has a minor impact on the total number of element updates with LTS, and thus only a minor impact on the run-time. At least in theory. In practice, the efficient implementation of LTS on a parallel computer is difficult, because the fine-grained dependencies between elements need to be dealt with. A trade-off between efficient implementation and

LTS speed-up is introduced in the clustered LTS scheme [16, 18]. Here, elements are grouped by time-step, forming time clusters. On the one hand, the theoretical speed-up decreases, because an element gets assigned the cluster's time-step which is typically lower than the element's maximum time-step. On the other hand, dependencies may be handled at cluster level, such that all elements within a cluster are updated in parallel.

The clustered LTS scheme [16, 18] is briefly reviewed in Section 8.1. The extension of LTS to support dynamic rupture is contributed in this thesis, which is discussed in Section 8.2. Moreover, a general data structure for LTS is introduced in Section 8.3 and load balancing is discussed in Section 8.4.

8.1 Clustered LTS

The family of ADER-DG schemes can be regarded as predictor-corrector schemes [55]. As reviewed in Section 3.2, the evolution of the PDEs in time is predicted in an element without accounting for its neighbours. Then, the degrees-of-freedom are corrected by inserting the prediction into the time-integrated semi-discrete DG scheme. In the corrector step, the time-integrated predictions of adjacent elements are required, see (3.35). Predictions are polynomials in time, see (3.32), and can be easily integrated over any interval within the time limits of the prediction. Therefore, LTS schemes are possible with arbitrary time-step ratios between adjacent elements [41].

In the clustered LTS scheme [16, 18] arbitrary time-step ratios are dropped in favour of multi-rate time-steps. Elements are grouped into time clusters, where the "zeroth" time cluster runs at the global minimum time-step and subsequent time clusters are integer multiples of previous time clusters. That is, the time clusters have time-steps

$$\forall c \in \mathbb{N} : \Delta t_c := r_c \cdot \Delta t_{c-1} \quad \text{with} \quad r_c \in \mathbb{N}, \tag{8.2}$$

and $\Delta t_0 := \Delta t_{\min}$. In practice, the rates r_i are often chosen to be equal. E.g. in a rate-2 scheme cluster time-steps are given by $\Delta t_c = 2^c \cdot \Delta t_{\min}$.

Classic domain decomposition is used on distributed memory systems: The dual graph of a tetrahedral mesh is plugged into a general purpose graph partitioner, e.g. ParMETIS [135], which returns a pairwise disjoint sets of elements $\mathcal{P}_r$ for each rank r. On each rank r, elements $m \in \mathcal{P}_r$ are grouped by time-step and then further sorted into ghost, copy, and interior layers. If element $m \in \mathcal{P}_r$ is part of the interior layer, then all of its

neighbouring elements m_f lie on the same rank, i.e. $\forall f \in [1\,..\,4] : m_f \in \mathcal{P}_r$. Copy layers contain elements with at least one neighbouring element that lies on another rank, i.e. $\exists f \in [1\,..\,4] : m_f \notin \mathcal{P}_r$. Elements in the ghost layer are itself not part of $\mathcal{P}_r$ but one of its neighbours is, i.e. $m \notin \mathcal{P}_r$ and $\exists f \in [1\,..\,4] : m_f \in \mathcal{P}_r$.

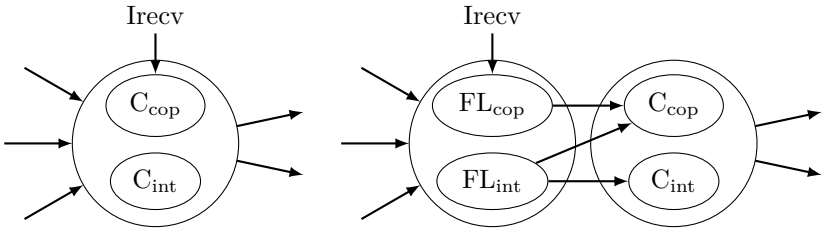
8.2 Dynamic rupture

In a dynamic rupture scenario, a subset of faces in the mesh is tagged as fault. On these faces, traction and slip rate are related via a friction law and the friction law is imposed weakly via non-linear numerical fluxes.

The implementation presented in Section 7.4 implicitly assumes that the two elements which share a fault face have the same time-step. In principle, one could create a scheme in which these elements have different time-steps. Such a scheme could be implemented using flux memory variables [41] and – ideally – nested quadrature rules. However, going one step back, the implementation in Section 7.4 does not require that *all elements* adjacent to the fault have to have the same time-step but only that each *pair of elements* that share a fault face have to have the same time-step, which is likely a minor constraint. Therefore, it is enforced that elements sharing a fault face have the same time-step.

Evaluating the friction law is expensive. For example for rate-and-state faults, the friction law is a differential-algebraic system of equations and involves solving non-linear equations, e.g. using the Newton-Raphson method [77]. Moreover, the prediction needs to be evaluated at space-time interpolation points, which is itself quite expensive. It is shown in Section 7.4 that the numerical flux for the two sides of an interface may be computed simultaneously, and as such the friction law evaluation and the space-time interpolation is only required once. However, such an approach requires a different parallelization strategy, as shall be outlined in the remainder of this section.

We first discuss the shared memory parallelization (with OpenMP): For linear numerical fluxes, the scheme is implemented using two parallel loops over elements, one for the prediction step and one for the correction step (these are also called local update and neighbouring update [16, 18]). The computation of the non-linear fluxes should not be part of the prediction step, because the prediction of the adjacent element is required. It should also not be part of the correction step, as the numerical fluxes need to be added to two elements, and as such race conditions might occur without explicit synchronisation (such as `#pragma omp critical`). Therefore, a



(a) Correction step. **(b)** Correction step with dynamic rupture.

Figure 15: Excerpt of the task graph for regular clustered LTS (left) and clustered LTS with support for dynamic rupture (right).

third parallel loop over fault faces, which we call the friction law loop, is introduced in between the prediction and the correction step. In the friction law loop, the tensors $\sum_{z=1}^{N^t} \gamma_z \mathcal{W}^-{}_{iqz}^{\ f}$ and $\sum_{z=1}^{N^t} \gamma_z \mathcal{W}^+{}_{iqz}^{\ f}$ are computed and stored. In the subsequent correction step, these tensors are used to apply (7.44) and (7.45) to the degrees of freedom.

In the distributed memory parallelization, we introduce copy and interior layers for fault faces: Let a face be identified by the tuple (k, m), where k and m are element numbers. Then, a face is part of the interior layer on rank r when $k \in \mathcal{P}_r$ and $m \in \mathcal{P}_r$. It is part of the copy layer on rank r when $k \notin \mathcal{P}_r$ or $m \notin \mathcal{P}_r$. Ghost layers are not required, because we compute the numerical fluxes separately on each rank in the case that a fault face lies on a partition boundary. (That is, space-time interpolation and friction law evaluation is required twice in this case.)

The scheduling of the friction law loop is shown in Figure 15. In the regular clustered LTS scheme, Figure 15a, the correction step for the copy layer and the interior layer (w.r.t. elements) share dependencies, with the exception of an "Irecv" task which signals that predictions of elements in the ghost layer have arrived. For dynamic rupture, we add two additional tasks for each time cluster: The friction law loop for the copy layer $\mathrm{FL}_{\mathrm{cop}}$ and the friction law loop for the interior layer $\mathrm{FL}_{\mathrm{int}}$, as shown in Figure 15b. The friction law inherits the dependencies of the correction step, whereas the correction step depends on the friction law loop. Note that $\mathrm{C}_{\mathrm{int}}$ only depends on $\mathrm{FL}_{\mathrm{int}}$, due to our definition of interior and copy layers for fault faces. Consequently, we may compute $\mathrm{FL}_{\mathrm{int}}$ and $\mathrm{C}_{\mathrm{int}}$ even before "Irecv" completed, i.e. before the ghost layer (w.r.t. elements) is received from another rank.

8.3 Data structure

Computational loops over elements require per-element data, e.g. degrees of freedom, the Jacobians (A^*), flux solvers ($\hat{\mathcal{A}}^{\pm}$), boundary conditions, face relation parameters (g_f, h_f), and also intermediate storage for derivatives or time-integrated degrees of freedom. We henceforth call each of these a *field* or a *variable*.

Classic data-structures are array of structures and structure of arrays. In the former the fields of an element are stored compactly, whereas in the latter each field is stored in a separate array. Structure of arrays often facilitates vectorisation and cache locality and is as such a popular choice for modern CPU architectures. However, standard operations such as memory allocation or field access become cumbersome, as one has to deal with multiple arrays. Therefore, containers have been developed which mimic an array of structures interface but use multiple arrays to store data, as in a structure of arrays layout [75, 128].

Structure of arrays is employed by Breuer [16]. Elements are organised hierarchically; they are first sorted by time cluster and then sorted by communication layer. Each array storing a field is ordered according to the element order, such that linear memory access is possible (with the exception of fields of neighbouring elements). The performance results [16, 18] indicate that the data structure is efficient for clustered LTS. However, the hierarchical data organisation requires additional boilerplate code:[1] For every time-cluster and communication layer combination, a separate object is created which stores a pointer for every field that points to the beginning of the cluster-layer combination in the field's array. A separate pointer for each field is required instead of a single offset because not every field is present on every communication layer. For example, the Jacobians are not required in the ghost layer.

We aim to combine the advantages of an automatic structure of arrays container with the efficient hierarchical structure developed for the clustered LTS scheme. Consequently, an *automatic tree-structured structure of arrays* container is developed in this thesis.

8.3.1 Automatic tree-structured structure of arrays

We recall that elements are sorted hierarchically, first by time cluster and then by communication layer. Trees are classic data structures to organise hierarchical data, therefore a tree structure is superimposed on a

[1] See code version `201511`, github.com/SeisSol/SeisSol.

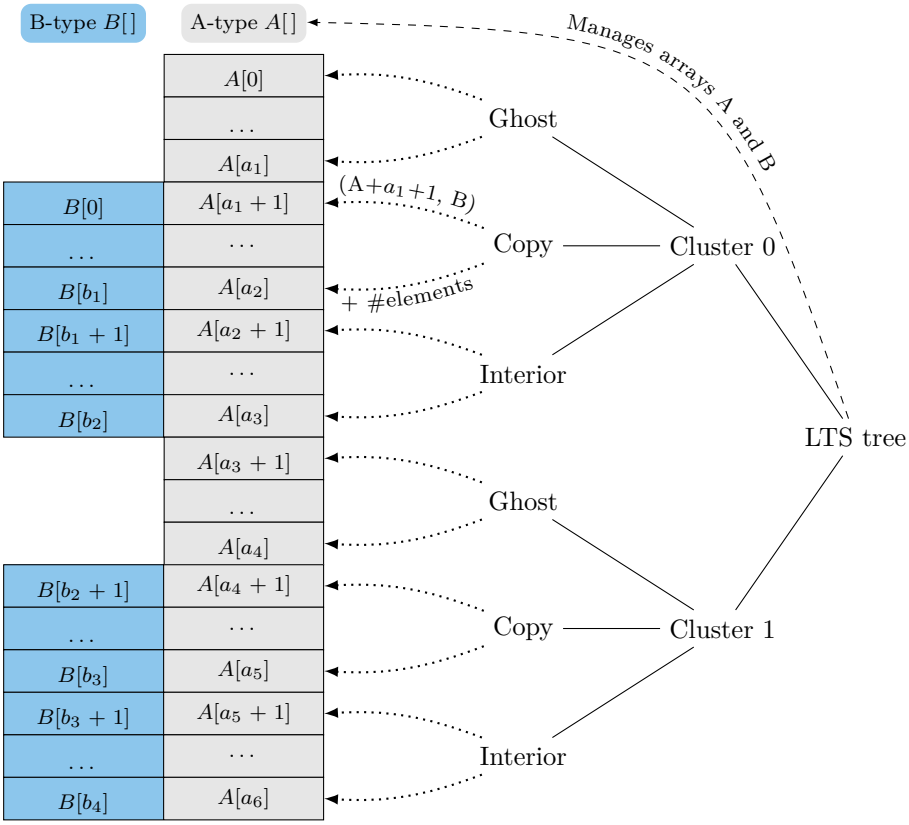


Figure 16: Fields A and B are organised in a tree (here for two time clusters). A contiguous chunk of memory is allocated for each field, which is managed by the root object. Elements are sorted, partitioned, and assigned to a leaf (ghost, copy, interior). The leaves store pointers to A and B, covering the fields of their subordinate elements.

structure of arrays data structure (henceforth called LTS tree). The basic concepts are illustrated in Figure 16: An LTS tree consists of any number of time-clusters, attached to the root node, and each time-cluster has the three children ghost, copy, and interior, which are the leaves of the tree. Each field is stored in one contiguous chunk of (virtual) memory, where the memory is managed by the root object. Elements are partitioned in time clusters and communication layers and are then assigned to the corresponding leaf. A leaf stores an iterator, that is, start address and number of elements, for each of its subordinate elements' fields. Note that a field may be *masked* if a field is not required in a communication layer. The field's array is still contiguous, though.

8.3.2 Handles

Fields are added at run-time to the LTS tree and are later referenced via *handles*. The handle is a close friend of the pointer, but with the difference that it is "dereferenced" by passing it to the root object or any of the leaves of the tree. The process is illustrated in Listing 5: Fields are defined in lines 2–4 and are added to the tree in lines 9–11. Several options may be defined. E.g. the field `dofs` is masked on the ghost layer and the array storing the field is allocated in high bandwidth memory, if

```
1  struct LTS {
2    Variable<double[tensor::Q::size()]> dofs;
3    Variable<CellLocalInformation>      cellInformation;
4    Variable<CellDRMapping[4]>          drMapping;
5  };
6
7  LTS lts;
8  LTSTree tree;
9  tree.addVar(lts.dofs, LayerMask(Ghost), 2097152, HighBandwidth);
10 tree.addVar(lts.cellInformation, LayerMask(), 1, HighBandwidth);
11 tree.addVar(lts.drMapping,  LayerMask(Ghost), 1,      Standard);
12 // Tree setup ...
13 tree.allocateVariables();
14 tree.touchVariables();
15
16 double (*dofs)[tensor::Q::size()];
17 dofs = tree.child(0).child<Copy>.var(lts.dofs);
```

Listing 5: Example usage of LTS tree.

available, and is aligned to 2 MiB pages. After tree setup and allocation in lines 12–13, the `dofs` handle is passed to a copy layer, see lines 16-17, in order to obtain a pointer to the first degrees of freedom stored in the layer. Before doing so, one should touch the fields as in line 14, which sets the fields to zero in parallel. Due to the first-touch policy, the latter step ensures that fields are distributed to all NUMA domains.

8.3.3 Kernel interface

A typical computational loop requires a subset of the fields of an element. An object which bundles these fields is advantageous as it may be easily passed to functions (an advantage of array of structures). Code generation is used for this purpose, as illustrated in Listing 6: In line 1, the preprocessor is invoked to create the class `LocalData`, which is used to select a subset of fields from the `LTS` structure storing the handles (see Listing 5). A loader object for an interior layer is created in lines 9–11. This object is used in line 15 to load the fields `dofs` and `cellInformation` for the element `e` from the interior layer. The object `data` contains references to the fields and may be used as in lines 16–17.

8.3.4 Data structure for dynamic rupture

For dynamic rupture we need to loop over fault faces and we also need to sort dynamic rupture faces by time cluster and communication layers, as explained in Section 8.2. Therefore, another tree-structured structure of arrays container is created for dynamic rupture. The tree for elements and the tree for fault faces is linked via pointer fields in each tree.

8.4 Load balancing

When global time-stepping is used, load is balanced by creating equally sized partitions of elements, e.g. using ParMETIS [135]. For local-time stepping, a simple and effective load balancing strategy is to weight an element by its number of updates [18]. E.g. for a scheme with constant rate r, elements in cluster c update

$$\frac{t_{\text{end}}}{\Delta t_c} = \frac{t_{\text{end}}}{\Delta t_{\min}} r^{-c} \tag{8.3}$$

```
LTSTREE_GENERATE_INTERFACE(LocalData, LTS, cellInformation, dofs)
// Creates the structure
// struct LocalData {
//   extract_type<decltype(LTS::cellInformation)>::type& X;
//   extract_type<decltype(LTS::dofs)>::type& X;
//   struct Loader { ... };
// }

auto& layer = tree.child(0).child<Interior>();
LocalData::Loader loader;
loader.load(lts, layer);

#pragma omp parallel for
for (unsigned e; e < layer.getNumberOfCells(); ++e) {
  auto data = loader.entry(cell);
  foo(data.dofs);
  bar(data.cellInformation);
  // ...
}
```

Listing 6: Illustration of the interface to the LTS tree used in computational loops.

times. So if $c_{\max}$ is the largest time cluster, then the weight

$$w_e := r^{c_{\max} - c_e}. \tag{8.4}$$

is assigned to element e in time cluster c_e.

For dynamic rupture, we get the computations on fault faces as additional load. A possible load-balancing strategy is to require that fault faces are also equally distributed, e.g. using multi-constraint partitioning [78]. Another possible strategy is to determine a fault face to element cost ratio, e.g. by measuring the average time of an element and the average time of a fault face. If that ratio is $p/q \in \mathbb{Q}^+$, then the integer weight assigned to element e is

$$w_e := (q + pn_e) r^{c_{\max} - c_e}, \tag{8.5}$$

where $0 \leq n_e \leq 4$ is the number of fault faces adjacent to element e.

Balancing load equally using one of the two mentioned strategies is only useful whenever the nodes crunch numbers equally fast. One would expect that the latter is true on a homogeneous supercomputer but it

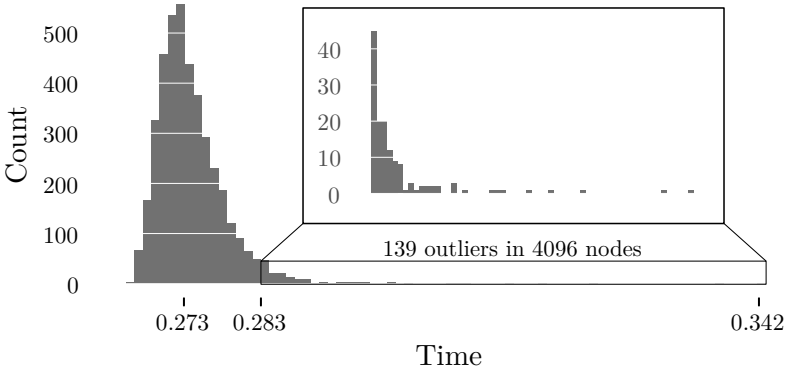


Figure 17: Distribution of execution time for an equal load on 4096 nodes of the Shaheen II supercomputer.

seems to be not necessarily true in reality: In an experiment conducted on the Shaheen II supercomputer, an equally sized computational load is run on 4096 nodes in parallel. Time is measured for the exact same code with the same number of iterations (i.e. same number of floating point operations). The distribution of measured times is shown in Figure 17. Data suggests that some nodes in the experiments perform significantly worse than the average node.

Already a single outlier is problematic: Assume that 4095 nodes require the average time of 0.273 s per unit load and that there is one node which requires 0.342 s per unit load. Assuming that load is equally balanced, the latter implies that the simulation requires 1.25× longer than if all 4096 nodes require the average time per unit load. On the contrary, assume that we assign zero load to the 139 outliers in Figure 17 and that the other 3957 nodes require the average time per unit load. Then, the simulation requires only 1.04× longer than if all 4096 nodes require the average time per unit load. Hence, letting 139 nodes run idle is faster than having a single "bad" node.

Consequently, we treat supercomputers as heterogeneous and pass node weights to the partitioner [79]. The computation of the node weights is straightforward: Similar to the experiment in Figure 17, we measure the

time it takes node r to compute a well-defined unit load. Say that time is t_r. Then the weight of node r is

$$p_r := \frac{1/t_r}{\sum_{i=1}^{\#\text{ranks}} 1/t_i}. \tag{8.6}$$

We note that the partitioner is called at run-time, using the PUML library [129], so the weights represent the state of the nodes at the beginning of the job.

8.5 Summary

We presented the extension of local time-stepping for dynamic rupture. Key developments are the extension of the LTS task graph and the automatic tree-structured structure of arrays container. The validation of the implementation is discussed in Section 10.4.

Furthermore, we presented load-balancing strategies. The impact of node weights is discussed in Section 12.1.

CHAPTER

9

easi: Rapid model setup

It is futile to measure time-to-solution only as the time spent on a supercomputer. From the viewpoint of a user, time-to-solution incorporates the whole pipeline from model setup to the post-processing of simulation output.

A common task in the setup of a simulation is to make certain input data available to the simulator. For example, Lamé parameters and density have to be set, a Dirichlet boundary condition which varies in space or time has to be defined, or an initial condition has to be prescribed.

The input data in seismology is rarely standardised. Instead, a variety of data sources and input formats are possible. Take the material parameters of an elastic medium as example, where a variety of community data sources and benchmarks exist:

— Layered media with layer-wise constant parameters [33],

— dynamic rupture benchmark problems specified with formulas or data given on uniform grids (in custom ASCII formats) [61, 62],

— the preliminary reference Earth model (PREM) [43], a global, one-dimensional, piecewise polynomial model of elastic properties,

— the Southern California Earthquake Center's Unified Community Velocity Model [140], which provides a unified API for several velocity models in southern California,

— the global CRUST1.0 model for elastic properties [99], which is given as ASCII files together with Fortran 77 code.

Evidently, none of these examples share a common input format but need to be treated on a case-by-case basis.

In many software packages for computational seismology, popular models like PREM are already included. For custom models, in particular three-dimensional models, one may either provide input data as uniform grid [27, 115, 119, 130] or one may resort to changing the source code of the software, which is for some packages advised in the documentation [27, 158].

Providing input data on a uniform grid seems to be a universal method, but the method has downsides: First of all, some models may produce an unnecessary large storage overhead, for example if the model contains only a few sharp material contrasts. Second, programming effort is not necessarily avoided, but imposed on the user, who needs to sample the input data on a uniform grid and write it to some file format. Third, file input and output might become a bottleneck in parallel simulations if not dealt with appropriately [130].

Changing the source code allows memory and compute efficient model initialisation. But, disadvantageous is that models are tightly coupled to the code and a model change might require re-compilation of the application.

In this chapter, an abstraction layer for input in simulation codes is proposed, which separates input and pre-processing of input from the underlying simulation software. The design goals of the abstraction are listed in the following:

1. Input data and numerical scheme are strictly separated,
2. the abstraction is powerful enough for a wide range of scenarios,
3. and a software implementation should be sufficiently efficient for large scale simulations.

An implementation of the abstraction layer is developed which is available to the public in the open-source library *easi*.[1]

9.1 Abstraction of input data

The abstraction of input data and the numerical scheme shall be strictly separated, as required by the first design goal. The only assumption we

[1] `https://github.com/SeisSol/easi`

make is that the numerical scheme requires input data at discrete points, which may be arbitrarily distributed in the simulation domain $\Omega \subset \mathbb{R}^d$. We require for discrete input data that some sensible interpolation function exists.

The nucleus of easi is the *component*. A component captures the various input methods that might be required in simulation codes, e.g. the evaluation of a formula, the interpolation of discrete data, or even the call to an external piece of code. In an abstract sense, a component is a tuple (f, χ), where f and χ are the functions

$$f : \mathbb{R}^m \to \mathbb{R}^n, \quad \chi : \mathbb{R}^m \times \mathbb{Z} \to \{0, 1\}. \tag{9.1}$$

We call f the map and χ the acceptance function. The map returns input data for a discrete point in the simulation domain, and the acceptance function indicates whether a component is valid for a point $\boldsymbol{x} \in \mathbb{R}^m$. The domain of the acceptance function includes an additional user-defined integer, on which the acceptance function may depend, too. A component is compatible with a simulation if $d = m$ and $\chi(\boldsymbol{x}, g) = 1$, for all $(\boldsymbol{x}, g) \in \Omega \times \Gamma$, where $\Gamma \subset \mathbb{Z}$.

The re-use of code shall be encouraged. Thus, it shall be possible to combine component A with another component B through composition, as long as the co-domain of f_A match the domain of f_B. For example, if a map for interpolating a structured grid is available as well as a map for linear affine transformations, then the composition of both maps can be used to rotate the data on the structured grid to another coordinate system. Composable components are called composite. A composite itself is a regular component (F, X). Additionally, the composite consists of a parent component (f, χ) and child components $(c_1, \tau_1), \ldots, (c_n, \tau_n)$. The map is given by

$$F(\boldsymbol{x}) = c_k(f(\boldsymbol{x})), \tag{9.2}$$

where k is the smallest integer such that $\tau_k(f(\boldsymbol{x}), g) = 1$. The acceptance function is

$$X(\boldsymbol{x}, g) = \chi(\boldsymbol{x}, g) \wedge \bigvee_{i=1}^{n} \tau_i(f(\boldsymbol{x}, g)). \tag{9.3}$$

9.2 Software architecture

The easi library is implemented using the Composite pattern [54]. Being a classic "Gang of Four" pattern, an explanation of the pattern is omitted. Instead, we briefly summarise some of the available components.

Composite. Implements composable components, i.e. (F, X) from Section 9.1. The base class of all subsequent components.

ASAGI, SCECFile. Implements D-linear interpolation on a D-dimensional Cartesian grid. Input files are either netCDF files, read with the scalable ASAGI library [130], or ASCII files in a custom format [62].

AffineMap, ConstantMap. Implements $f(\boldsymbol{x}) = A\boldsymbol{x} + \boldsymbol{t}$, where $A \in \mathbb{R}^{n\times m}$ and $\boldsymbol{t} \in \mathbb{R}^n$. ConstantMap is a special AffineMap with $A = 0$.

DomainFilter. Points may be filtered using simple geometric entities. For example, the acceptance function of the sphere $(\boldsymbol{c}, r) \in \mathbb{R}^m \times \mathbb{R}^+$ is $\chi(\boldsymbol{x}, g) = (\|\boldsymbol{x} - \boldsymbol{c}\| \leq r)$.

EvalModel. Component which evaluates another easi model within an easi model.

FunctionMap. Just-in-time compiled function with simplified C-like syntax. Uses a fork of the ImpalaJIT library [50].[2]

GroupFilter. Implements acceptance function $\chi(\boldsymbol{x}, g) = (g \in G)$ for the user-defined set $G \subset \mathbb{Z}$.

PolynomialMap. Implements one-dimensional polynomials of the form $f(x_1) = \sum_{i=0}^{N} a_i x_1^i$.

Special. Allows to call custom C++-user functions.

Switch. Special composite component. The input vector $\boldsymbol{x}$ is partitioned into M sub-vectors $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_M$ and passed on M child components. I.e. $F(\boldsymbol{x}) = (c_1(\boldsymbol{x}_1), \ldots, c_M(\boldsymbol{x}_M))$.

9.3 Input format

Models are specified in configuration files following the YAML specification [10]. We present the file format by the example of the PREM, shown in Listing 7.

The top-level component first transforms the three-dimensional input vector to a normalised radius in line 6, as specified in the PREM [43]. The children of the top-level component are specified in lines 7–42, whose input is the normalised radius from line 6. PREM is given in terms of several layers. A separate model is specified for each layer using domain

[2] `https://github.com/uphoffc/ImpalaJIT`

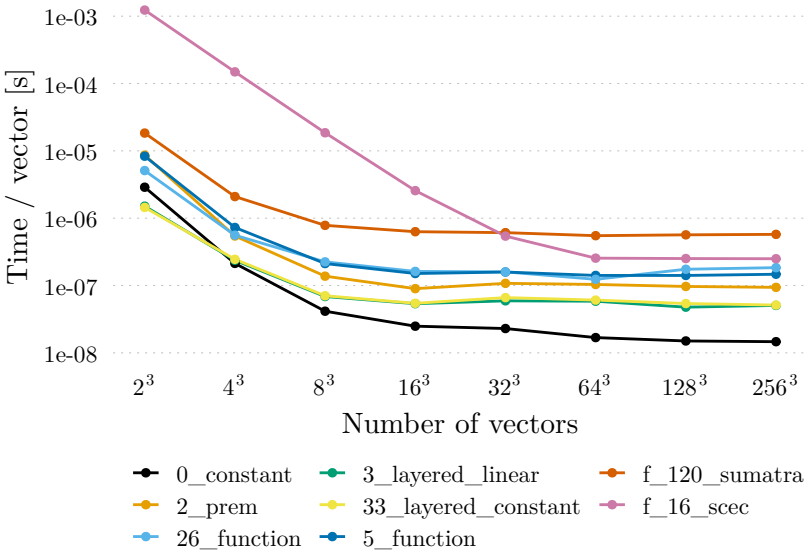


Figure 18: Scaling analysis of the easi benchmark suite. The time per vector decreases with increasing number of vectors. Figure adapted from [155].

filters. In each layer, a polynomial map or a constant map is specified, where the coefficients are copied from the initial publication [43, Table I].

PREM is specified in terms of (ρ, v_p, v_s), i.e. density, P-wave velocity, and S-wave velocity. Therefore, the final step in the model is to convert this parameter set to (ρ, μ, λ) with proper units. The conversion is implemented using a just-in-time compiled function for each output parameter, see lines 19–21. Note that the latter component is annotated with the label `&PREM2SeisSol`, which is later referenced (line 31 and line 42), such that the conversion function needs to be written only once.

9.4 Performance

Evaluating a model requires a tree traversal as well as virtual method calls. In order to reduce the model evaluation overhead, data parallelism is implemented. That is, multiple input parameters are passed to the model evaluation function simultaneously, i.e. an input matrix, instead of

```
# Preliminary reference Earth model
# Adam M. Dziewonski and Don L. Anderson, Physics of the Earth and
# Planetary Interiors 25, pp. 297--356, 1981.
!FunctionMap
map:
  r: return sqrt(x*x + y*y + z*z) / 6371000.0;
components:
  # Inner core 0--1221.5 km
  - !AxisAlignedCuboidalDomainFilter
    limits:
      r: [0, 0.19172814314864228536]
    components: !PolynomialMap
      map:
        rho:  [-8.8381, 0, 13.0885]
        vp:   [-6.3640, 0, 11.2622]
        vs:   [-4.4475, 0,  3.6678]
      components: &PREM2SeisSol !FunctionMap
        map:
          lambda: return 1000000000.0*rho*(vp*vp - 2.0*vs*vs);
          mu:     return 1000000000.0*rho*vs*vs;
          rho:    return 1000.0*rho;
  # Outer core 1221.5--3480 km
  - !AxisAlignedCuboidalDomainFilter
    limits:
      r: [0.19172814314864228536, 0.54622508240464605243]
    components: !PolynomialMap
      map:
        rho:  [ -5.5281,  -3.6426,  -1.2638, 12.5815]
        vp:   [-13.5732,   4.8023,  -4.0362, 11.0487]
        vs:   [       0,        0,        0,       0]
      components: *PREM2SeisSol
  # ...
  # Ocean 6368--6371 km
  - !AxisAlignedCuboidalDomainFilter
    limits:
      r: [0.99952911630827185685, 1.0]
    components: !ConstantMap
      map:
        rho:  1.020
        vp:   1.450
        vs:   0.000
      components: *PREM2SeisSol
```

Listing 7: Excerpt of the input file for the PREM.

a single input vector. Components receive input points in matrix form and may therefore optimise (and especially vectorise) their evaluation code.

A suite of benchmarks, which is based on material models used in practice, is evaluated in the following.[3] An experiment on a single core of an Intel Skylake Platinum 8174 processor shows the relation between the number of input vectors and the time per vector, see Figure 18. Data suggests, that using 16^3–32^3 vectors per model evaluation is often sufficient, whereas using few vectors leads to sub-optimal performance.

The most expensive benchmark in the collection is `f_120_sumatra`. About 9.7 s are required in order to evaluate 16.8 million input vectors on a single core. In SeisSol, easi needs to be queried only at the time of initialisation, and the number of input vectors scales linearly with the number of elements or fault faces. As large simulations typically require several hours, the time spent in easi is therefore acceptable.

9.5 Impact

In the 2017 release of SeisSol [158], the model initialisation code is spread over thousands of lines of Fortran code and is tightly integrated with the internals of the software. The latter code has been completely removed in favour of calls to the easi library. Easi models are now routinely used in geophysical research with SeisSol [152, 153].

[3] Available on `https://github.com/SeisSol/easi/tree/201910/examples`.

CHAPTER

10

Benchmarks and verification

A numerical scheme and its implementation must be verified. Therefore, verification methods and exercises are discussed in the following.

10.1 Convergence tests

The ADER-DG scheme for the elastic and the viscoelastic wave equation achieve high-order accuracy in space and time. This property is verified in convergence tests for plane-wave problems in homogeneous media in the original publications [39, 81].

10.1.1 Plane-wave IVP

Plane waves have the form

$$p_p(\boldsymbol{x}, t) := a_p \exp\left(i(\omega t - \boldsymbol{k} \cdot \boldsymbol{x})\right), \tag{10.1}$$

where $\omega \in \mathbb{C}$ is the angular frequency, $\boldsymbol{k} \in \mathbb{R}^3$ the wave vector, $i^2 = -1$, and $\boldsymbol{a} \in \mathbb{C}^{\#\text{quantities}}$ is the initial amplitude vector. Plugging (10.1) in the system of PDEs (3.1) yields the eigenproblem

$$(k_d A_{pq}{}^d - i E_{pq}) a_q = \omega a_p. \tag{10.2}$$

We call the matrix on the left-hand side the *plane-wave operator*, and we denote its eigenvalue-eigenvector pairs with $(\omega_j, \boldsymbol{r}^j)$.

From (10.2) we see that a solution of the form (10.1) requires that the angular frequency is an eigenvalue of the plane-wave operator and that the initial amplitude vector is the corresponding eigenvector. In general, such a solution might be complex, which we like to avoid in the convergence tests. Hence, we actually look for solutions with the following structure

$$q_p(\boldsymbol{x}, t) := \frac{1}{2}\left(a_p \exp\left(i(\omega t - \boldsymbol{k}\cdot\boldsymbol{x})\right) + \overline{a_p}\exp\left(-i(\overline{\omega} t - \boldsymbol{k}\cdot\boldsymbol{x})\right)\right), \quad (10.3)$$

which have the nice property that

$$q_p(\boldsymbol{x}, t) = \mathrm{Re}\left(a_p \exp\left(i(\omega t - \boldsymbol{k}\cdot\boldsymbol{x})\right)\right). \quad (10.4)$$

If angular frequency and initial amplitude vector are an eigenpair of the plane-wave operator, then (10.3) is a solution, too.

The plane-wave solutions require a domain of infinite extent, which is quite impractical for a simulation code. However, under the condition that $\boldsymbol{k} = \boldsymbol{m}\pi/S$, $\boldsymbol{m} \in \mathbb{Z}^3$ and $S \in \mathbb{R}^+$, the plane-wave is periodic on the cube $\Omega := [-S, S]^3$.

In summary, we define the periodic plane-wave IVP as following:

$$\begin{aligned}
\frac{\partial q_p}{\partial t} + A_{pq}{}^d \frac{\partial q_q}{\partial x_d} &= E_{pq} q_q, \quad \forall (\boldsymbol{x}, t) \in (-S, S)^3 \times (0, t_{\mathrm{end}}),\\
q_p(S, x_2, x_3, t) &= q_p(-S, x_2, x_3, t), \quad \forall (x_2, x_3, t) \in [-S, S]^2 \times (0, t_{\mathrm{end}}),\\
q_p(x_1, S, x_3, t) &= q_p(x_1, -S, x_3, t), \quad \forall (x_1, x_3, t) \in [-S, S]^2 \times (0, t_{\mathrm{end}}),\\
q_p(x_1, x_2, S, t) &= q_p(x_1, x_2, -S, t), \quad \forall (x_1, x_2, t) \in [-S, S]^2 \times (0, t_{\mathrm{end}}),\\
q_p(\boldsymbol{x}, 0) &= \sum_{j \in \mathcal{J}} \mathrm{Re}\left(r_p{}^j \exp\left(-i\pi S^{-1} \boldsymbol{m}\cdot\boldsymbol{x}\right)\right), \quad \forall \boldsymbol{x} \in [-S, S]^3,
\end{aligned} \quad (10.5)$$

where $S \in \mathbb{R}^+, t_{\mathrm{end}} \in \mathbb{R}^+$, and $\boldsymbol{m} \in \mathbb{Z}^3$. Multiple waves may be superimposed due to linearity, therefore the sum over a set $\mathcal{J} \subset [1\,..\,\#\mathrm{quantities}]$ is taken.

IVP (10.5) has the analytic solution

$$q_p(\boldsymbol{x}, t) = \sum_{j \in \mathcal{J}} \mathrm{Re}\left(r_p{}^j \exp\left(i(\omega_j t - \pi S^{-1} \boldsymbol{m}\cdot\boldsymbol{x})\right)\right). \quad (10.6)$$

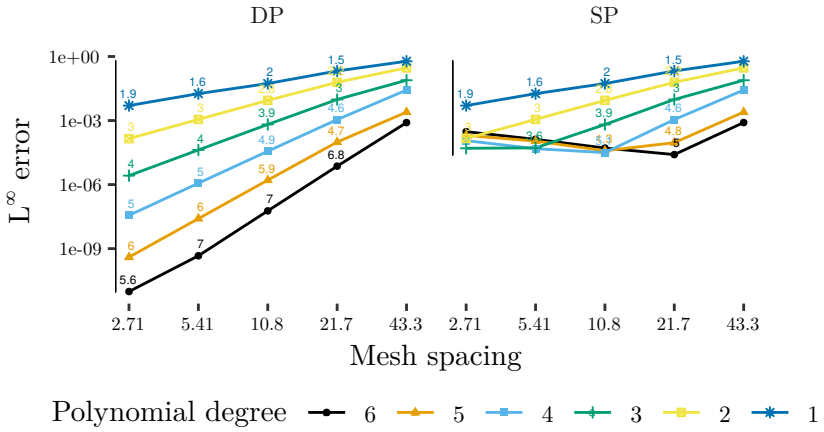


Figure 19: Periodic plane-wave problem for the elastic wave equation. Shown is the L^∞ error and the empirical convergence orders of the σ_{yz} component of the stress tensor.

10.1.2 Empirical convergence order

For the periodic plane-wave problems, the error is expected to decay exponentially with with mesh refinement, i.e. the error in the L^p norm behaves as following:

$$E^h_{L^p} = \left\| q_p - q^h_p \right\|_{L^p} \leq C h^{N+1}, \tag{10.7}$$

where C is a constant independent of h, N is the polynomial degree, and h is the maximum diameter of the circumsphere of a tetrahedron, in the following called mesh spacing.

Following Dumbser et al. [39], we construct a sequence of meshes for an experimental convergence test with mesh spacings $h_1, h_2, \ldots$ such that $h_i = h_{i-1}/2$. The empirical convergence orders are computed with

$$\mathcal{O}_{L^p} = \log_2 \left(\frac{E^{h_{i-1}}_{L^p}}{E^{h_i}_{L^p}} \right). \tag{10.8}$$

For the elastic wave equation, we set $S = 50, \boldsymbol{k} = (1,1,1) \cdot \pi/50, t_{\text{end}} = 100\sqrt{3}$, and the material parameters are set to $\lambda = 2, \mu = 1, \rho = 1$, following Dumbser et al. [39]. A P-wave travelling in the direction of $\boldsymbol{k}$ and an S-wave travelling in the opposite direction are imposed. The L^∞ error and the empirical convergence orders for the quantity σ_{yz} are

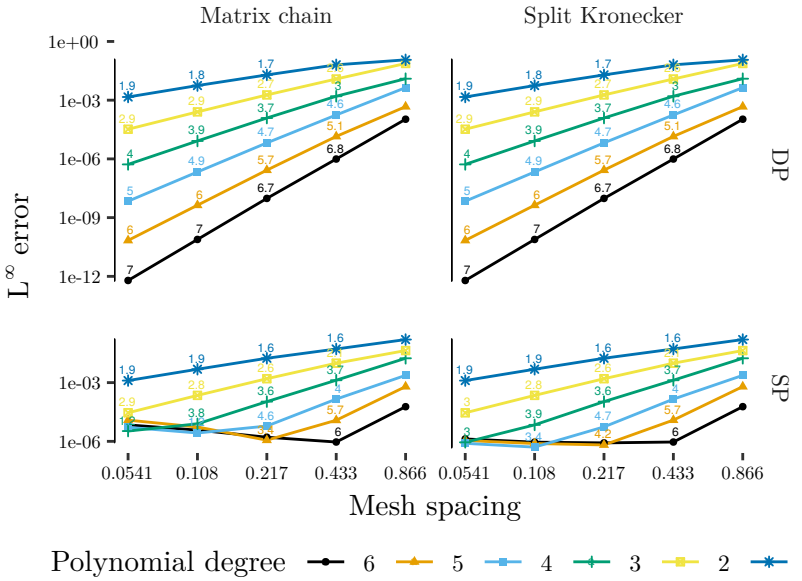
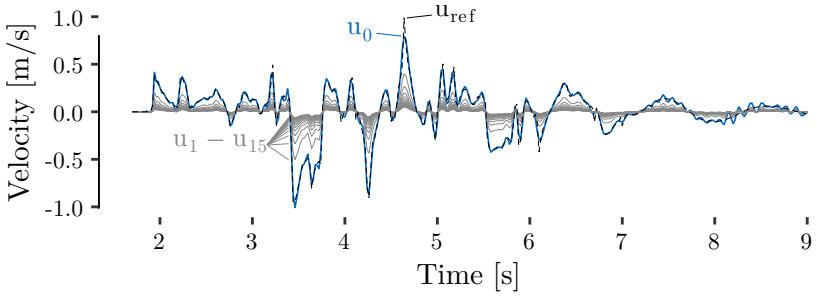


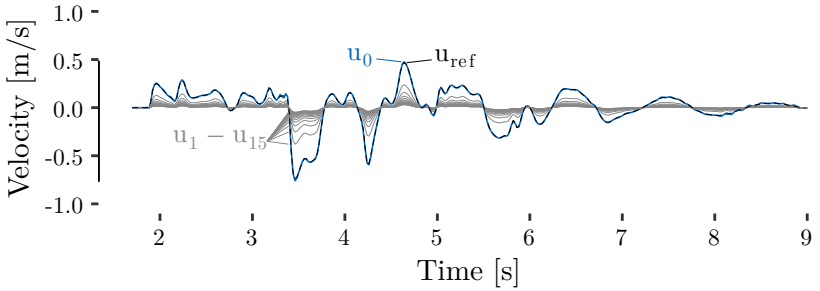
Figure 20: Periodic plane-wave problem for the viscoelastic wave equation. Shown is the L^∞ error and the empirical convergence orders of the σ_{yz} component of the stress tensor.

shown in Figure 19. We observe that the empirical convergence orders closely match $N + 1$ in double precision. In single precision arithmetic, the minimum attainable error is limited, which is also observed by Breuer et al. [21].

For the viscoelastic wave equation, we set $S = 1, \boldsymbol{k} = (1, 1, 1) \cdot \pi, t_{\text{end}} = 0.1$, and the material parameters are set to $\lambda = 2, \mu = 1, \rho = 1, Q_P = 20, Q_S = 10$, following Käser et al. [81]. A P-wave travelling in the direction of $\boldsymbol{k}$ and an S-wave travelling in the opposite direction are imposed. The L^∞ error and the empirical convergence orders for the quantity σ_{yz} are shown in Figure 20. We observe that the empirical convergence orders closely match $N + 1$ in double precision for both implementations described in Section 7.3. In single precision arithmetic, the minimum attainable error is again limited.



(a) LOH.1 (elastic)



(b) LOH.3 (viscoelastic)

Figure 21: Ensemble simulation of the LOH.1 and the LOH.3 benchmark. Figures show the first component of the 9-th receiver (cf. Tables 7 and 8). The ensemble simulations reduce the amplitude as expected by 1/2, 1/3, 1/4, etc. Figure 21a from [157].

10.2 Layer over halfspace: Ensemble simulations

Many features are not tested by the periodic plane-wave problem, such as material interfaces, the free surface boundary condition, and kinematic rupture sources. A suite of more involved benchmarks is available in the SeISmic MOdeling Web INterfacE (`www.sismowine.org`). Here, model descriptions and reference solutions are provided, and accuracy levels are defined.

The layer over halfspace problem (LOH.1) is chosen as a benchmark problem for ensemble simulations. As specified in the benchmark,[1] a $[-26\,\mathrm{km}, 32\,\mathrm{km}] \times [-26\,\mathrm{km}, 32\,\mathrm{km}] \times [0\,\mathrm{km}, 34\,\mathrm{km}]$ simulation domain is used. The first kilometre in depth, $z \in [0\,\mathrm{km}, 1\,\mathrm{km}]$, is called the layer with a free surface boundary condition at $z = 0\,\mathrm{km}$, and the remainder, $z \in (1\,\mathrm{km}, 34\,\mathrm{km}]$ is called the halfspace. All but the free surface boundary are set to absorbing. Wave velocities in the layer are $c_p = 4000\,\mathrm{m\,s^{-1}}$, $c_s = 2000\,\mathrm{m\,s^{-1}}$, and the density is $\rho = 2600\,\mathrm{kg\,m^{-3}}$. In the halfspace, one sets $c_p = 6000\,\mathrm{m\,s^{-1}}$, $c_s = 3464\,\mathrm{m\,s^{-1}}$, and $\rho = 2700\,\mathrm{kg\,m^{-3}}$.

A point source is buried at $(0\,\mathrm{km}, 0\,\mathrm{km}, 2\,\mathrm{km})$. The moment-rate history is $\partial M_{12}/\partial t = \partial M_{21}/\partial t = M_0\, t/T^2 \exp(-t/T)$ and zero otherwise, where $T = 0.1\,\mathrm{s}$ and $M_0 = 10^{18}\,\mathrm{N\,m}$. In order to test ensemble simulations, we vary the seismic moment by the factor 1/2, 1/3, 1/4, etc. That is, M_0 is replaced by $M_s = M_0/(1+s)$ for the s-th simulation in the ensemble. The reference solution for these problems is given by scaling the reference solution of SISMOWINE with the same factor.

Figure 21a shows one of the seismograms recorded for an ensemble simulation in single precision with an ensemble size of 16 and enabled local time-stepping. The first simulation (with moment M_0) closely matches the reference solution. As expected, the other simulations in the ensemble have their amplitude reduced by the factors $1/2, 1/3, \ldots, 1/16$.

The benchmark is executed for a total of eight configurations: Tested are global-time stepping and local-time stepping, single precision and double precision, and single simulation and ensemble simulation ($S = 8$ for DP and $S = 16$ for SP). All configurations are run with order 6 on a mesh with 1,116,326 elements. Mesh refinement is used in the vicinity of the source.

We quantify the misfits by computing the three-component Envelope Misfit (EM) and Phase Misfit (PM) [92, 93]. The maximum EM and PM for all configurations is shown in Table 7. We observe that all misfits are well below the 5 % limit, which classifies the numerical solution as highly accurate according to the benchmark description.

10.3 Layer over halfspace: Viscoelasticity

The layer over halfspace problem for the viscoelastic wave equation (LOH.3) is almost identical to the layer over halfspace problem for the elastic wave equation (LOH.1). The same mesh as in Section 10.2 is used and the setup

[1] See `www.sismowine.org/model/WP2_LOH1.pdf`.

Table 7: LOH.1: Maximum misfit observed in different configurations (see text). The envelope misfit (EM) and the phase misfit (PM) are given in per cent. All receivers are placed at the free surface.

rec.	Position [m] x	y	u_1 [%] EM	PM	u_2 [%] EM	PM	u_3 [%] EM	PM
1	0	693	0.8	0.3	0.5	0.0	0.3	0.0
2	0	5543	1.3	0.2	0.3	0.0	0.1	0.0
3	0	10392	1.3	0.2	0.4	0.0	0.2	0.0
4	490	490	0.8	0.3	1.0	0.4	0.6	0.1
5	3919	3919	1.2	0.1	1.4	0.1	0.8	0.1
6	7348	7348	0.7	0.1	0.7	0.1	1.0	0.2
7	577	384	0.6	0.2	0.9	0.3	0.5	0.1
8	4612	3075	1.1	0.1	1.3	0.1	0.6	0.1
9	8647	5764	1.0	0.2	1.2	0.2	1.0	0.2

is almost identical. We only need to introduce quality factors for the layer, which are $Q_P = 40$ and $Q_S = 120$, and quality factors for the halfspace, which are $Q_P = 69.3$ and $Q_S = 155.9$. The P- and S-wave velocities are defined at 2.5 Hz (the phase velocity is frequency dependent).

The same eight configurations as in Section 10.2 are run for the sixth order scheme with three relaxation mechanisms, and misfits are computed for each receiver. The maximum misfits are shown in Table 8, which are all well below the 5 % limit.

A visual impression of a seismogram is shown in Figure 21b. The effect of attenuation is clearly visible in the reduced amplitudes compared to Figure 21a. Furthermore, the amplitudes in the ensemble are correctly reduced according to the seismic moment.

10.4 The Problem, version 16

Verification of the numerical scheme and implementation of dynamic rupture is difficult, as analytical solutions, even for simple problems, are absent [34]. Only a solution exists for a shear crack propagating at a constant velocity [89], but in this setting the generated waves do not interact with the “fault”.

Table 8: LOH.3: Maximum misfit observed in different configurations (see text). The envelope misfit (EM) and the phase misfit (PM) are given in per cent. All receivers are placed at the free surface.

rec.	Position [m] x	y	u_1 [%] EM	PM	u_2 [%] EM	PM	u_3 [%] EM	PM
1	0	693	1.5	0.2	0.3	0.0	0.1	0.0
2	0	5543	0.8	0.2	0.1	0.0	0.1	0.0
3	0	10392	0.7	0.2	0.2	0.0	0.1	0.0
4	490	490	1.5	0.2	1.6	0.2	0.7	0.1
5	3919	3919	0.6	0.2	0.8	0.2	0.6	0.1
6	7348	7348	0.4	0.1	0.5	0.1	1.1	0.2
7	577	384	0.9	0.1	1.5	0.2	0.5	0.1
8	4612	3075	0.6	0.2	0.7	0.1	0.4	0.1
9	8647	5764	0.7	0.2	0.6	0.2	1.1	0.2

Community benchmarks have been developed to allow the comparison of various implementations and numerical schemes, including finite difference, finite element, and discontinuous Galerkin schemes [61, 62]. Forty benchmarks are available to test various aspects of a dynamic rupture code. All benchmarks are called "The Problem" together with a version number, e.g. TPV5 or TPV104. The numerical scheme used in SeisSol has been tested for many of the TPV problems [122].

We verify that local-time stepping for dynamic rupture works as intended for the TPV16 problem [158]. The fault is planar and vertical, with an extent of 48 km by 19.5 km. A linear slip-weakening friction law is used and the initial stress field is heterogeneous. In order to test LTS, we discretise the fault with varying resolutions: A 50 m resolution is used in the area of nucleation, surrounded by a transition zone with 200 m resolution and a 150 m resolution is used elsewhere, cf. Figure 22. Fault faces lie in four time clusters, where 99.8 % of the faces lie in the clusters $2\Delta t_{\min}, 4\Delta t_{\min}$, and $8\Delta t_{\min}$. The full setup is available at Zenodo [158].

The numerical solution, shown exemplary in Figure 23, is in excellent agreement with the established dynamic rupture codes FaultMod [8] and SPECFEM3D [27]. Local time-stepping and global time-stepping give almost indistinguishable solutions.

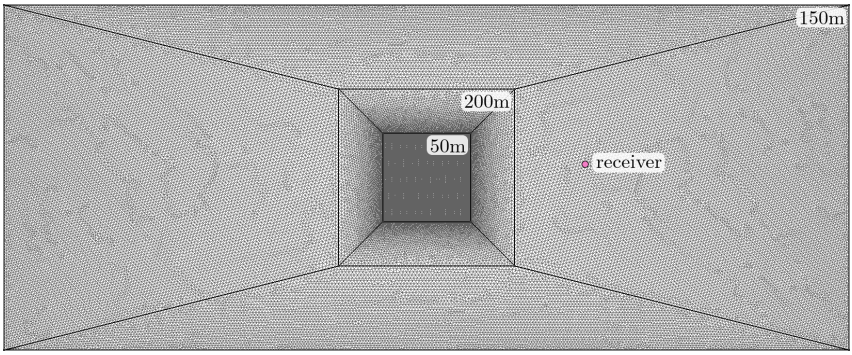


Figure 22: Illustration of the fault discretisation for the TPV16 problem. Fault faces are distributed in four time clusters. Figure from [158].

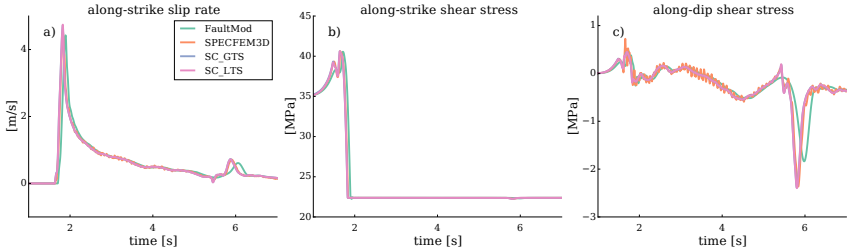


Figure 23: Comparison of the numerical solution at the location marked in Figure 22 (9 km depth and 9 km along-strike). SeisSol is denoted with the prefix SC. Figure from [158].

10.5 Single precision vs. double precision

SeisSol is commonly used with double precision. However, as we shall see in Chapter 11, using single precision may reduce the computation time by up to a factor to two. Single precision quickly reaches a precision limit. However, single precision is sufficient w.r.t. envelope and phase misfit in the LOH.1 and LOH.3 benchmark problems, which is found by Heinecke et al. [65] for the LOH.1 problem, too. So as long as the floating point error is insignificant, one might prefer single precision as one can simulate a finer mesh in the same computation time.

A potential problem with single precision might be the limited value range for cells containing point sources. As these cells are refined, the discrete Dirac delta becomes larger and larger such that at some point a value larger or equal 2^{128} is produced, which cannot be represented in single precision arithmetic. However, one should be able to deal with these cells by either switching to double precision in the cells containing point sources, or by rescaling the quantities.

Albeit the potential advantages of single precision, we use double precision in this thesis. The reason is that we only recently started to investigate single precision and have not verified the use of single precision for dynamic rupture problems. Nevertheless, performance results in Chapter 11 are given for single and double precision, as single precision might be beneficial in future work.

CHAPTER

11

Single and multi node performance

The performance and scalability of the implementation of ADER-DG is evaluated thoroughly for three different computing systems, which are introduced in Section 11.1. Many results in this chapter are obtained using YATeTo and the implementation described in Chapter 7. However, earlier results of the author for the flux matrix decomposition are also included [158]. For these results a predecessor of YATeTo is used [156, 158]. In order to quickly distinguish implementations, we refer to the *Baseline (BL)* version [16, 64], the *Shaking Corals (SC)* version [158], and the *YATeTo (YT)* version [157] throughout this chapter.

Single-node performance is analysed in Sections 11.2 and 11.3. There, we use a performance proxy application, which calls the exactly same kernels as SeisSol, but uses random initial data and elements have random face neighbours and random face relations. Previous work shows that the performance of the proxy predicts the performance of actual simulations very well if global-time stepping is used [16, 64, 156, 158].

We return to the layer over halfspace benchmark in Section 11.4 and evaluate the performance of multi-node simulations in different configurations. In Section 11.5 the scalability of the implementation of viscoelastic attenuation is investigated.

11.1 Computing systems

Three different computing systems with Intel architectures are used in this thesis: Dual-socket *Haswell* nodes (HSW), single-socket *Knights Landing* nodes (KNL), and dual-socket *Skylake SP* nodes (SKX). A summary of architectural details is tabulated in Table 9.

The machine balance B_m, measured in byte per floating-point operation, and the arithmetic intensity of a code A_c, measured in floating-point operation per byte, give an indication of the possible performance of an algorithm on a computing system. That is, if the product of both is smaller than one, the performance is limited by the available memory bandwidth. On the contrary, if the product is larger than one, the performance is limited by the computing system's peak performance. The general trend seems to be a decreasing machine balance [60, Figure 3.2], such that one needs to increase the number of (useful) flops per byte transferred in order to fully exploit modern architectures.

Caches are small chunks of memory which lie in between the main memory and the CPU. On HSW and SKX, three cache layers are available, ordered from large and slow (L3) to small and fast (L1). (KNL has two cache layers.) Due to increased bandwidth of caches, the machine balance increases when data gets closer to the core. That is, the performance depends on the cache level in which data resides.

Data may or may not be present in one of the two or three cache levels. Modelling where data resides in a real application is non-trivial as one needs to account for the cache line replacement policy (which might not be disclosed by the vendor) and also for the physical memory addresses (caches are commonly set-associative). However, the opposite is much easier to model, i.e. where data does not reside. For example, the degrees of freedom are too large too stay in low level caches from one loop over elements to another loop over elements. And, starting from degree $N = 5$, the 48 flux matrices cannot simultaneously reside in the level 2 cache of any of the three architectures; we return to this point in Section 11.2.

We conclude this section with a short discussion of peak performance: The theoretical peak performance is computed as the product of the maximum number of flops per cycle, the number of cores, and the core frequency. While the former two are well specified, the precise core frequency seems to become more and more fuzzy with each generation. For KNL, the only official document known to the author states that one shall add 100 MHz for the all-tile turbo frequency, but subtract 200 MHz for "high-AVX instruction frequency" [31]. It does not become clear what

Table 9: Overview of computing systems used in experiments. Data assembled from [24, 29–32] or based on measurements if indicated.

	HSW	**KNL**	**SKX**
Model [Intel Xeon]	E5-2697v3	Phi 7250	Platinum 8174
Sockets	2	1	2
Cores	14	68	24
L1d or L1i [KiB core^{-1}]	32	32	32
L2 (private) [KiB socket^{-1}]	14×256	34×1024[a]	24×1024
L3 (shared) [MiB socket^{-1}]	35	N/A	33
Base frequency [GHz]	2.6	1.4	2.7 / 3.1
All-core AVX2 [GHz]	2.2[b]	N/A	3.0 / 3.2
All-core AVX512 [GHz]	N/A	1.5	2.5 / 2.8
DP peak perf. [TFLOP s^{-1}]	1.0	3.3	3.8 / 4.3
DRAM bandwidth [GB s^{-1}]	108[c]	90	204[c]
HBM bandwidth [GB s^{-1}]	N/A	490	N/A
Machine bal. [B FLOP^{-1}]	0.11	0.027 / 0.15[d]	0.054 / 0.048

[a] KNL has 34 tiles. Each tile consists of 2 cores that share an L2 cache.
[b] Capped at 2.2 GHz due to the computing centre's energy policy.
[c] Measured with STREAM benchmark. Corresponds to $> 79\,\%$ of the theoretical memory bandwidth.
[d] Assuming data resides in high bandwidth memory (HBM).

distinguishes a high-AVX instruction frequency from a not-so-high-AVX instruction frequency. Thus, we calculate the peak performance based on the optimistic guess of 1.5 GHz. For SKX, a specification document for the 8174 model does not seem to exist, such that frequencies are taken from the computing centre's website [24]. Moreover, the nodes are operated in either the 205 W or the 240 W mode [24], with base frequencies 2.7 GHz or 3.1 GHz, respectively.

11.2 Flux matrix decomposition

The results in this section were first presented by Uphoff et al. [158]. We discuss the performance of the SC version.

In Section 7.1 we discussed the large cache requirements of the 48 flux matrices. The latter may be reduced by sorting vertices [132] (leading to only 16 flux matrices), by using prefetching [64], or by using the flux matrix decomposition.

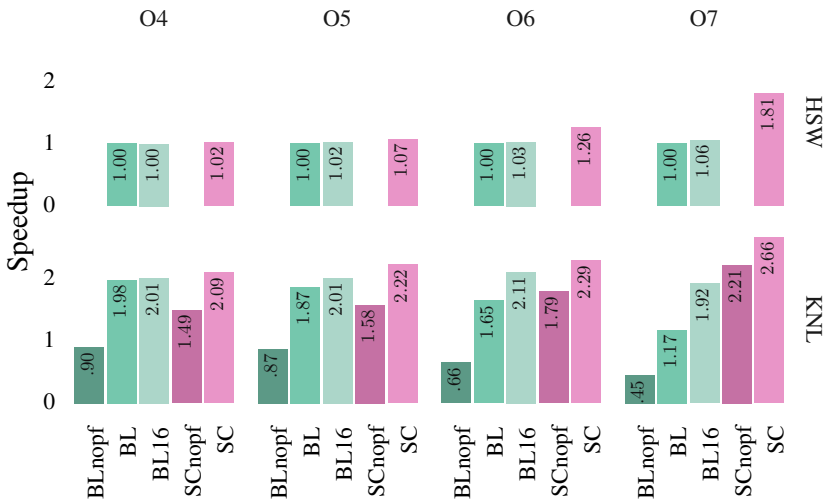


Figure 24: Figure shows the speed-up of the neighbouring surface integral computation obtained with flux matrix decomposition in the Shaking Corals (SC) version. The standard baseline version requires 48 flux matrices, but we also show a modified version which only accesses 16 flux matrices (BL16). Speed-ups are computed w.r.t. BL on HSW for every order. Suffix *nopf* denotes disabled prefetching. Figure from [158].

We evaluate five implementation variants using SeisSol's performance proxy application.[1] The first variant is the baseline variant without prefetching (BLnopf). The second baseline variant (BL) uses an advanced software prefetching scheme, in which the flux matrices as well as the neighbouring degrees of freedom are prefetched [64]. In the third variant, we emulate the best case that only 16 flux matrices need to accessed by restricting the face relations to 16 possible variants (BL16). The flux matrix decomposition is used in the fourth variant (SCnopf). Prefetching is also advantageous for the fourth variant, leading to a fifth variant with prefetching (SC). The prefetching scheme in SC is simpler in comparison to BL, as only the neighbouring degrees of freedom are prefetched.

[1] The proxy application is available on `https://github.com/SeisSol/SeisSol/tree/master/auto_tuning/proxy`. Check out tag `201703` for the SC version.

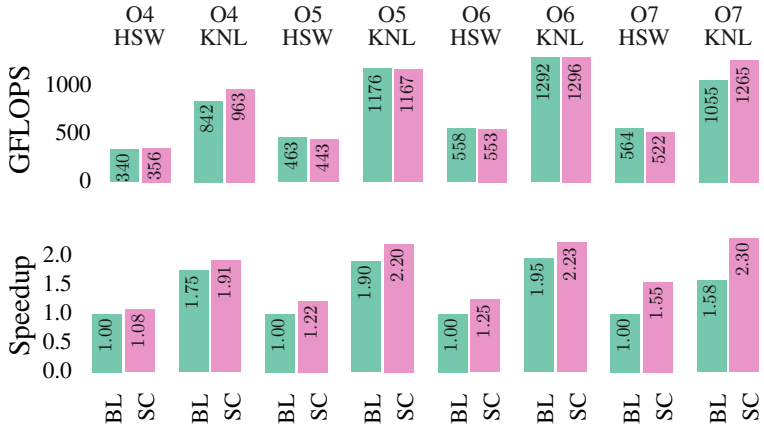


Figure 25: Single node speed-up due to flux matrix decomposition. Data is based on runs with 100000 random elements. Figure from [158].

Figure 24 shows the speed-up in the computation of the neighbouring surface integral. On HSW, almost no speed-up can be seen for orders 4 and 5.[2] For orders 6 and 7 we see a speed-up of up to 1.81 for SC, but almost no speed-up for BL16. The speed-up for $\mathcal{O}7$ is interestingly larger than the operation count ratio of 1.36, cf. last row in Table 4. Therefore, we can assume that caching effects are important even though a large L3 cache is present.

On KNL, we see that BL16 is up to 1.64 times faster than BL. However, the flux matrix decomposition delivers superior performance, and is up to 2.27 times faster than BL or 1.39 times faster than BL16. Note that prefetching of the neighbouring degrees of freedom is still important, which becomes evident when comparing SC and SCnopf.

The impact of the neighbouring surface integration on the total runtime is shown in Figure 25. For $\mathcal{O}7$ a total speed-up of 1.55 is achieved on HSW and 1.46 on KNL. The speed-up for $\mathcal{O}4$ is 1.08 on HSW and 1.09 on KNL. Up to 56 % of peak performance is achieved on HSW and up to 40 % of peak performance is achieved on KNL. Note that the performance of BL and SC is about equal in some instances. This is not a contradiction to

[2] Order refers to convergence order, as verified in Section 10.1, i.e. order $\mathcal{O} = N + 1$.

the observed speed-up, as the flux matrix decomposition requires a smaller number of floating point operations.

The results in this section demonstrate that the smaller working set of the flux matrix decomposition and the smaller number of operations has a clear performance benefit, and has been already been adopted by another group [65]. However, one is left to wonder whether one has to rely on empirical evidence or if one could also predict the performance benefits of the flux matrix decomposition or similar optimisations. The latter question is in particular important if one wants to automatise the implementation of numerical schemes even more, and leave such implementation details to a compiler. Precise modelling of which cache lines stays in which cache level at which point in time is likely difficult or maybe infeasible. But falsification is simple: We know that the 48 flux matrices cannot stay in L2 cache simultaneously once the polynomial degree gets too high (cf. Table 4). Therefore, falsification may be a valuable tool when exploring the space of implementations.

11.3 Ensemble simulations

In this section, the performance of single and ensemble simulations is evaluated using the performance proxy. We use the metric degrees of freedom per second, as the implementations we compare for the viscoelastic wave equation require different numbers of hardware and non-zero flops, such that comparing flops becomes meaningless. Nevertheless, the flop-metrics are of interest as the hardware flops show how well we utilise the hardware, and the non-zero flops show the possible gains of exploiting sparsity. Therefore, the results of this section are tabulated in Appendix A for reference, including non-zero flops, hardware flops, and median absolute deviation of hardware flops.

11.3.1 Elastic wave equation

The implementation of ADER-DG for the elastic wave equation is tested for orders 2–7, single precision and double precision, single and ensemble simulations, and on the systems KNL and SKX (in 240 W mode). The results are shown in Figure 26.

First of all, we observe that the YATeTo version (YT) reproduces the performance of the Shaking Corals (SC) version. In all cases, YT is as fast as SC or slightly faster.

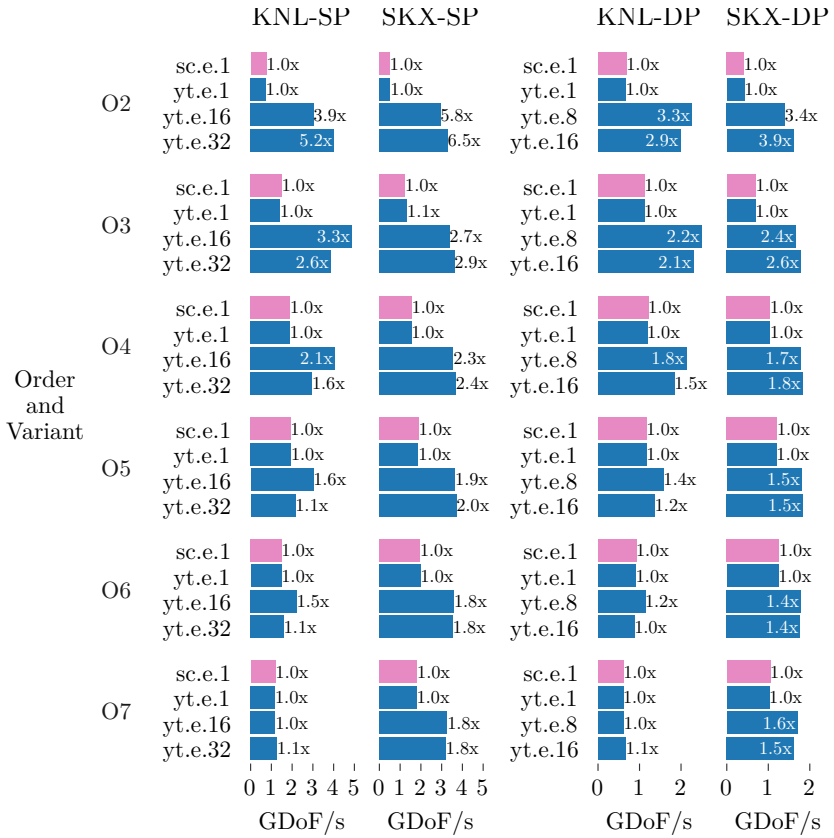


Figure 26: Elastic wave equation: We compare the performance in degrees of freedom per second of several implementations for orders 2–7, single precision (SP) and double precision (DP), and two different computing systems (KNL and SKX 240 W). Numbers beside bars show the increase in throughput compared to the SC version. The implementation variants are encoded as ⟨version⟩.e.⟨ensemble size⟩. Data from [157].

Second, we observe that ensemble simulation increase the throughput by 1.1×–3.9× in double precision (DP) and 1.1×–6.5× in single precision (SP). Ensemble simulations give the highest speed-ups for low order simulations, which is explained by the higher arithmetic intensity and by the perfect fit of vector width to leading tensor dimension (which is the ensemble size) [20]. But note that also for orders with high arithmetic intensity, such as $\mathcal{O}6$ or $\mathcal{O}7$, we get a decent increase in throughput on SKX.

Throughput increase does not tell us how well an implementation utilises the hardware. We therefore discuss performance data at the example of $\mathcal{O}7$ and $\mathcal{O}2$ on SKX-DP (cf. Appendix A): For $\mathcal{O}7$, the performance proxy achieves $2.2\,\mathrm{TFLOP\,s^{-1}}$ or about 52 % of peak performance for a single simulation. That is, a 512 bit fused multiply-add is computed every cycle. However, many zero flops are computed, as exploiting sparsity does not reduce the time-to-solution. In terms of the non-zero flops metric, about 16 % are achieved. The $\mathcal{O}7$ ensemble simulation with ensemble size 8 has a non-zero efficiency of 27 % and a hardware efficiency of 35 %. For $\mathcal{O}2$, we observe 1.7 % and 7.9 % for a single simulation, and 6.4 % and 7.4 % for ensemble size 16.

We generally observe over all experiments that ensemble simulations reduce the hardware efficiency but increase the non-zero efficiency. This is due to increased use of dense × sparse matrix multiplications, which reduce the time-to-solution but have a lower arithmetic intensity.

11.3.2 Viscoelastic wave equation

Three implementation variants are discussed in Section 7.3: The matrix chain (MC) implementation, the unsplit Kronecker (UK) implementation, and the split Kronecker (SK) implementation. We tested these implementations on the SKX system (in 205 W mode) for orders 2–7, single precision and double precision, and single and ensemble simulations. The results are shown in Figures 27 and 28.

We first discuss single simulations: The Kronecker product implementations are always faster than the matrix chain implementation, up to 2×. Generally speaking, the Kronecker variants becomes more effective with increasing number of mechanisms and decreasing order. At high orders, the split Kronecker variant and the unsplit Kronecker variant perform about equally well. But for low orders, the split Kronecker variant performs noticeably better than the unsplit Kronecker variant.

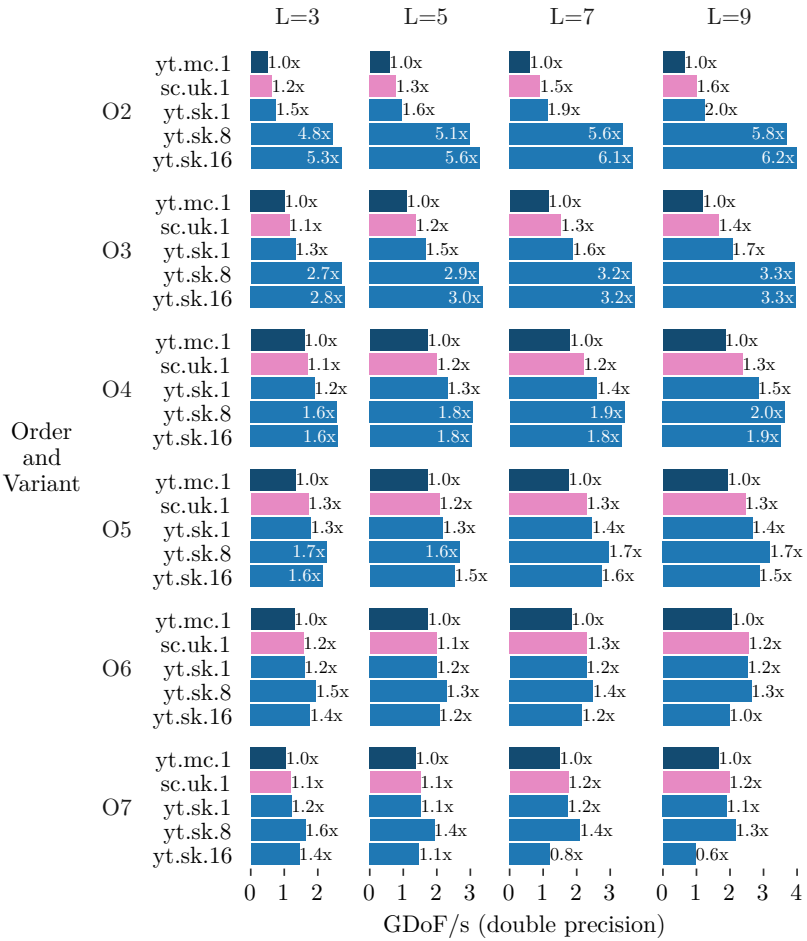


Figure 27: Viscoelastic wave equation: We compare the performance in double precision degrees of freedom per second of several implementations for orders 2–7, several numbers of relaxation mechanisms ($L = 3, 5, 7, 9$). Numbers beside bars show the increase in throughput compared to the matrix chain implementation. All tests are run on SKX 205 W. The implementation variants are encoded as ⟨version⟩.⟨variant⟩.⟨ensemble size⟩.

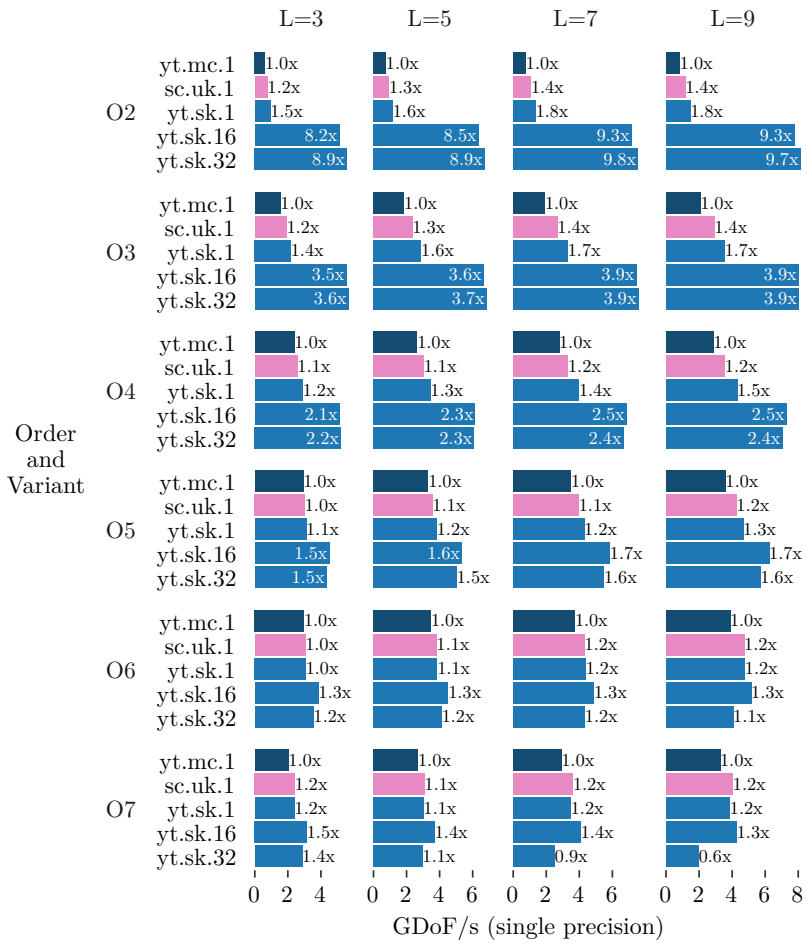


Figure 28: Viscoelastic wave equation: Single precision. Cf. caption of Figure 27.

Ensemble simulations increase the throughput for the viscoelastic wave equation, too. The only exception being an ensemble size of 16 for $\mathcal{O}7$ in DP or an ensemble size of 32 for $\mathcal{O}7$ in SP.

In terms of peak performance, we exemplary present the split Kronecker variant for $\mathcal{O}7$ and 3 mechanisms on SKX-DP (cf. Appendix A): A hardware efficiency of 48 % and a non-zero efficiency of 19 % is obtained for a single simulation, and a hardware efficiency of 27 % and a non-zero efficiency of 26 % is obtained for ensemble size 8.

For 9 mechanisms and $\mathcal{O}7$, the hardware efficiency of the split Kronecker variant decreases to 37 % and the non-zero efficiency to 18 % for a single simulation. Nevertheless, the number of degrees of freedom per second increases from $1.2\,\mathrm{GDoF\,s^{-1}}$ to $1.9\,\mathrm{GDoF\,s^{-1}}$. The reason for this behaviour lies in the structure of the viscoelastic wave equation, in which the number of floating point operations does not grow linearly with the number of quantities. Furthermore, for the matrix chain variant we see an increase from $1.1\,\mathrm{GDoF\,s^{-1}}$ to $1.7\,\mathrm{GDoF\,s^{-1}}$, when moving from 3 mechanisms to 9 mechanisms for $\mathcal{O}7$. Here, the sub-linear growth of operations is automatically exploited by equivalent sparsity patterns (cf. Section 6.2.1 and Figure 10).

11.4 Layer over halfspace

We return to the layer over halfspace benchmark problems from Sections 10.2 and 10.3. Single precision and double precision are tested, as well as global time-stepping and local time-stepping. Eight nodes are used for the LOH.1 benchmark and 17 nodes for the LOH.3 benchmark.

The results in Tables 10 and 11 demonstrate that the measured performance agrees well with the performance proxy for global time-stepping, even though multiple nodes are used and the proxy operates on a random mesh. The maximum deviation is 10 % for LOH.1 and 2 % for LOH.3. Moreover, the speed-ups of ensemble simulations are closely reproduced. That is, 1.5× for LOH.1-DP, 2× for LOH.1-SP, 1.2× for LOH.3-DP, and 1.3× for LOH.3-SP.

The speed-up of local time-stepping (in comparison to global time-stepping) ranges from 2.1×–2.3× and is a bit lower than the theoretical speed-up of 2.6×.

Table 10: LOH.1 benchmark performance using the sixth order scheme on 8 nodes (SKX 240 W). The simulation time is normalised with the ensemble size.

P	LTS	Ensemble	P_{NZ}	P_{HW}	% Proxy	t_{sim} [min]
DP	No	1	569	1772	91	38.7
DP	No	8	839	1148	92	26.6
DP	Yes	1	508	1580	81	16.8
DP	Yes	8	699	957	77	12.4
SP	No	1	894	3449	90	24.6
SP	No	16	1766	2629	96	12.6
SP	Yes	1	768	2962	77	11.1
SP	Yes	16	1460	2174	80	5.9

Table 11: LOH.3 benchmark performance using the sixth order scheme, three relaxation mechanisms, and the split Kronecker variant on 17 nodes (SKX 205 W). The simulation time is normalised with the ensemble size.

P	LTS	Ensemble	P_{NZ}	P_{HW}	% Proxy	t_{sim} [min]
DP	No	1	649	1481	98	40.2
DP	No	8	805	902	99	32.6
DP	Yes	1	553	1263	83	18.3
DP	Yes	8	652	731	80	15.6
SP	No	1	1248	3409	98	20.9
SP	No	16	1604	1932	100	16.3
SP	Yes	1	1009	2757	79	10.0
SP	Yes	16	1290	1555	80	7.9

11.5 Strong scaling

Lastly, the scalability on distributed memory systems is investigated. The 1,116,326 element mesh from Sections 10.2 and 10.3 is the object of study, but scaled to 256 nodes. That is, we scale down to 4361 elements per node or 91 elements per core on average. We discuss the different implementation variants of the viscoelastic wave equation. Note that only the wave

Table 12: Comparison between the matrix chain variant and the unsplit and split Kronecker variants for global and local time-stepping. Speed-up is calculated for a single node. Multiply speed-up with the number of nodes and the parallel efficiency in order to obtain the speed-up on multiple nodes. Scaling tests are run on SKX 205 W nodes. Each run is repeated at least four times and the lowest wall time is used to compute speed-ups and peak efficiencies.

		Parallel efficiency [%]									
Variant	Speed-up	1	2	4	8	16	32	64	128	192	256
yt.mc.gts	1×	100	98	98	96	95	93	89	85	81	79
yt.mc.lts	1.791×	100	104	99	94	90	84	77	67	60	58
sc.uk.gts	1.172×	100	99	98	97	95	91	87	82	78	77
sc.uk.lts	1.973×	100	104	98	95	90	82	74	67	61	58
yt.sk.gts	1.168×	100	99	99	99	98	98	97	94	93	91
yt.sk.lts	2.625×	100	104	101	98	96	91	84	76	70	71

propagation part is tested; scalability of realistic application scenarios, including dynamic rupture, is discussed in Chapter 12.

Scaling tests are run on SKX 205 W nodes (in double precision). Each run is repeated at least four times and the best run is considered. Table 12 shows the parallel efficiencies of the matrix chain (MC), the unsplit Kronecker (UK), and the split Kronecker (SK) variants for global and local time-stepping. For global time-stepping, we observe that SK scales best and achieves a parallel efficiency of over 90 % on 256 nodes. MC and UK fall below 90 % starting from 64 nodes. For local time-stepping, SK performs best, too. SK falls below 90 % starting from 64 nodes whereas MC and UK fall below 90 % starting from 32 nodes.

The best-case scenario is listed in Table 12. But for high node counts, performance shows a strong run-to-run variability as visible in Figure 29. A notable exception is also one out of four runs on 2 nodes for the UK variant, which takes 1.23× longer than the best run. The sources of variability are subject to speculation and further experiments are necessary to understand them.

The speed-ups in Table 12 show that the Kronecker variants are faster than the matrix chain variant for GTS, and that the two Kronecker variants are about equally fast (on a single node). We expect so from Section 11.3.2. Unexpected is, that SK is much faster than UK for LTS. We

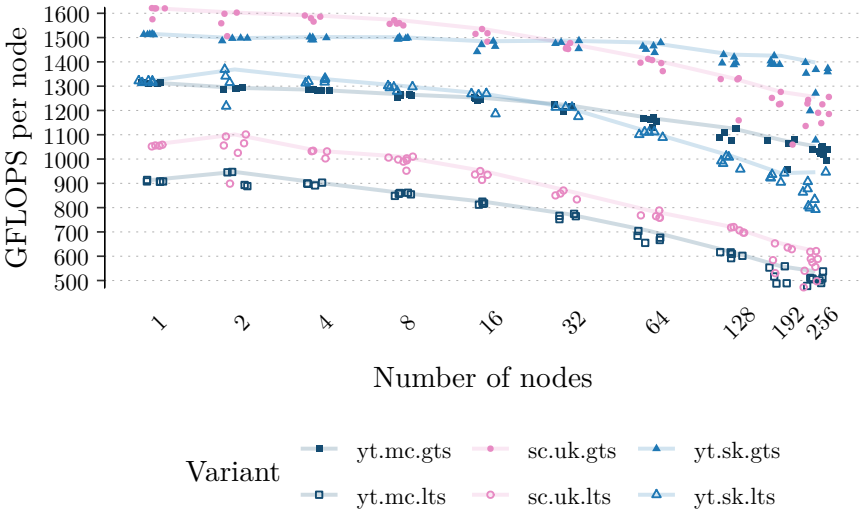


Figure 29: Strong scaling performance in hardware GFLOP s^{-1} per node. LTS performs worse than GTS for all three implementation variants. Furthermore, a strong run-to-run variability is observed for 256 nodes.

observe in Figure 29 that performance decreases are worse for UK and MC than for SK.

Comparing the kernels in SK and the kernels in UK and MC leads to the hypothesis that most of the performance is "lost" in the neighbour update for LTS. In the neighbour update the time-integrated degrees-of-freedom for the four neighbours of an element have to be loaded from memory (see (7.30) and second line of (7.8)). When LTS is enabled, a neighbour element might have a larger time-step, such that the time-integrated degrees-of-freedom are obtained by integrating the Taylor expansion of the neighbour element in time [16] (please also see (3.33)). In this case, $N \cdot \binom{N+3}{3}$ floating point numbers have to be loaded in addition for every quantity. So for an $N = 5$ scheme in double precision, at least an additional

$$\frac{5 \cdot 56 \cdot 8\,\mathrm{B}}{204 \cdot 10^9\,\mathrm{B\,s^{-1}}} \approx 11\,\mathrm{ns}, \tag{11.1}$$

are required per quantity. (Estimated with STREAM bandwidth from Table 9.) In the unsplit schemes (MC and UK) 27 quantities are loaded

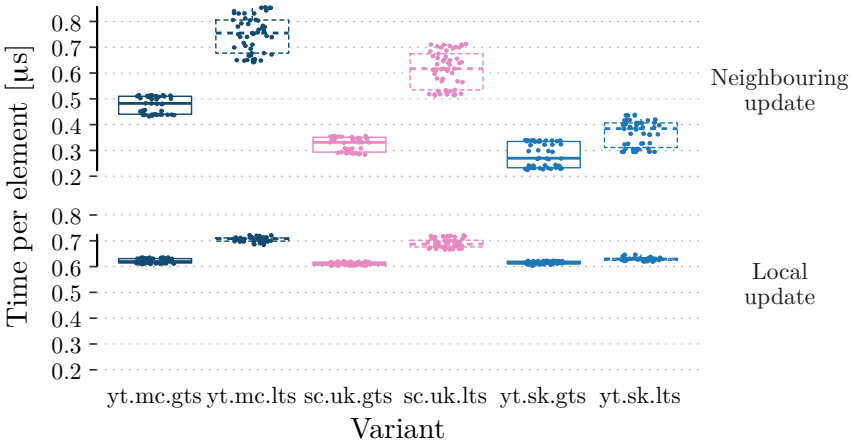


Figure 30: Figure shows the average time per update step per node, estimated with SeisSol's inbuilt linear regression analysis.

leading to an additional 297 ns per neighbouring update per node. In the split scheme (SK), only 9 quantities are loaded, such that the loading overhead is reduced to 99 ns. Note that only two additional flops are required per floating point number, i.e. the integration of the Taylor expansion is memory bound with arithmetic intensity below one.

Evidence confirming the neighbour update hypothesis is found with the internal monitoring tools of SeisSol: A stopwatch is placed around the three major parallel loops, which measures the time of the local update, the neighbouring update, and the friction law evaluation. Time is stored as a sample, annotated with the loop trip count. The pairs of loop trip count and time are regarded as random sample and the relation between the two is modelled with linear regression. The slope of the resulting linear model is interpreted as the average time per element per node and the intercept is interpreted as the average startup and shutdown time of a parallel loop.

Figure 30 shows the slopes of the regression analysis measured in the strong scaling tests. Only GTS on a single node is excluded in Figure 30, because in this particular case we only have a single unique loop trip count and therefore the regression analysis is underdetermined and unreliable. We observe that the major deviation between GTS and LTS stems from the neighbouring update: The difference w.r.t. to the median slopes is 273 ns for MC, 286 ns for UK, and 115 ns for SK. (For the local update

we have 89 ns for MC, 75 ns for UK, and 15 ns for SK.) So, the order of magnitude of additional time per element as well as the ratio between the unsplit and split variants support the hypothesis that additional loads in the neighbouring update are the major cause of additional time per element.

In retrospect, loading 27 quantities is unnecessary for the matrix chain and the unsplit Kronecker variants, and these variants could be optimised as well. However, loading only 9 quantities is obvious in the split Kronecker formulation and not so obvious in the other two formulations, such that the split Kronecker formulation is preferable.

11.6 Conclusions

The flux matrix decomposition reduces cache usage and reduces the number of floating point operations for high orders. Experiments on HSW and KNL confirm that a speed-up of about 1.5× is achieved compared to the baseline implementation.

Ensemble simulations increase the throughput considerably, in particular for $\mathcal{O}2$–$\mathcal{O}4$ schemes, and might be a valuable tool for certain applications such as probabilistic seismic hazard analysis. However, given the restrictions associated with ensemble simulations, these are not universally applicable. We showed that the kernels generated with YATeTo are efficient for single and ensemble simulations. Moreover, we demonstrated in Chapter 7 that the kernels for single and ensemble simulations are generated from the same DSL code. Hence, we infer that it is sustainable to support both operating modes simultaneously.

The speed-up of the split Kronecker implementation in comparison to the matrix chain implementation ranges from 1.1× to 2.0× for single simulations. For typical production scenarios with $\mathcal{O} \geq 4$ and three relaxation mechanisms, the speed-up is at most 1.3×. However, the scalability for the split Kronecker variant is superior to the matrix chain variant for GTS as well as for LTS. The unsplit Kronecker variant performs about equally well as the split Kronecker variant for GTS. But for LTS, the parallel efficiency of the unsplit Kronecker variant falls off.

CHAPTER

12

Supercomputing

Quantitative accuracy analysis of high order ADER-DG schemes suggests that one needs less than two elements per wavelength to accurately resolve seismic wave propagation [82]. So for resolving high frequency seismic waves, the number of degrees of freedom quickly goes into the billions.

For example, assume a seismologist wants to accurately resolve frequencies of 5 Hz in a cube with a side length of 100 km for the viscoelastic wave equation with three relaxation mechanisms. Given an S-wave velocity of $3\,\mathrm{km\,s^{-1}}$, the shortest wavelength is 600 m. Therefore, the seismologist chooses an edge length of about 300 m and the $\mathcal{O}6$ scheme. He is going to require about

$$5 \cdot \frac{(100 \cdot 10^3\,\mathrm{m})^3}{(300\,\mathrm{m})^3} \cdot \binom{5+3}{3} \cdot 27 = 280 \cdot 10^9$$

degrees of freedom.[1] Evidently, he requires a supercomputer.

In the hypothetical scenario, time-steps are in the order of milliseconds and simulating a few seconds of wave propagation is feasible using global time-stepping [16, 67]. However, once realistic topography or bathymetry, material layers, and faults are introduced, slivers lead to sharply declining time-step sizes.

[1] A cube is divided in 5 tetrahedra and 27 quantities are needed.

Tiny time-steps are a challenging issue in practice: In Section 12.1 we study an elastic dynamic rupture model of the 2004 Sumatra-Andaman earthquake with 111 billion degrees of freedom. Estimates of the rupture duration are over 8 min [138] but the smallest time-step is 151 µs. The tiny time-step only affects a few slivers, but with global time-stepping the 111 billion degrees of freedom need to be updated 3.3 million times. We thus evaluate the extension of local time-stepping for dynamic rupture and investigate the scalability on SuperMUC Phase 2, Shaheen II, and Cori. Details of these supercomputers are listed in Table 13.

Table 13: Overview of supercomputers used in this work. Rmax is the maximum LINPACK performance and Rpeak is the theoretical peak performance of the system. Data assembled from [112].

	M	**S**	**C**	**NG**
Name	SuperMUC Phase 2	Shaheen II	Cori	SuperMUC-NG
Node type	HSW	Haswell[a]	KNL	SKX
Number of nodes	3072	6144	9152	6372
Rmax [PFLOP s^{-1}]	2.8	5.5	14.0	19.5
Rpeak [PFLOP s^{-1}]	3.6	7.2	27.9	26.9

[a] Dual-socket Intel Xeon E5-2698 v3. Similar to HSW, but with an additional two cores per socket, 40 MiB L3, and 2.3 GHz base frequency.

Recently, the SuperMUC-NG supercomputer was put into operation, which is ranked ninth in the November 2019 TOP500 list [112]. In comparison to SuperMUC Phase 2, the number of nodes has more than doubled and the LINPACK performance increased by 7×. Scalability and performance for large-scale simulations on this system is tested in Section 12.2. Here, we use a model of the 2010 Darfield earthquake with a viscoelastic rheology as object of study.

12.1 The 2004 Sumatra-Andaman earthquake

The 2004 Sumatra-Andaman earthquake was an extreme event in every respect: Few earthquakes exceeded its moment magnitude of M_W 9.1–9.3 [138]. The rupture extended over a length of 1020–1500 km and lasted for 480–600 s [138]. The earthquake caused the Indian Ocean tsunami,

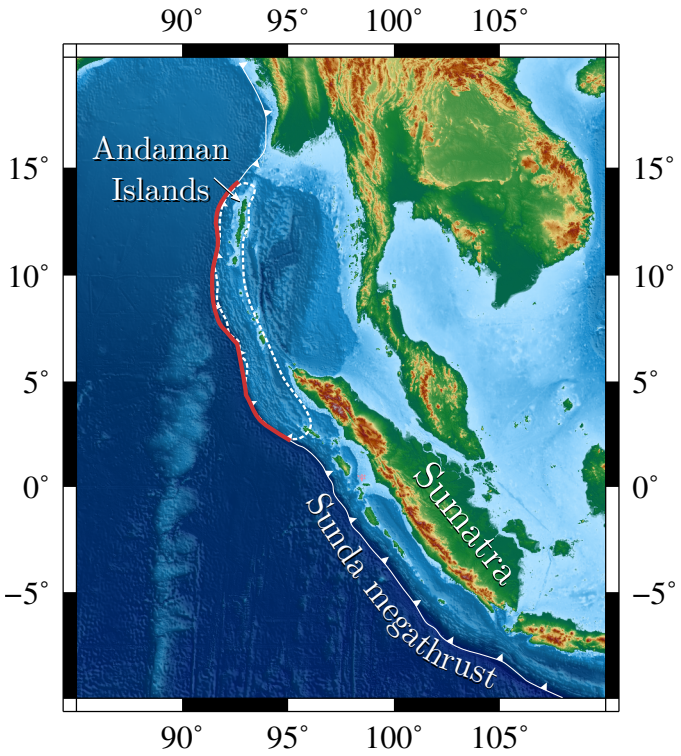


Figure 31: The Sunda megathrust ruptured over a length of 1300–1500 km during the 2004 Sumatra-Andaman earthquake. The rupture extent is enclosed in the dashed line. Figure adapted from [138], topography and bathymetry from [59].

for which run-up heights of up to 50 m were measured in the province of Aceh [137].

The Sumatra earthquake occurred on parts of the Sunda megathrust, see Figure 31. Its proximity to northern Sumatra and the Andaman Islands turned it into a devastating human catastrophe, with an estimated death toll of 230,000 people [137]. Over 167,000 people were killed in Indonesia alone, and tens of thousands of deaths were reported in Sri Lanka, mainland India, and Thailand, even though the first waves arrived 2–3 h after tsunamigenesis in the latter three countries.

An earthquake and tsunami in such extent came unexpected, and even several years after the event there is no full agreement on the earthquake's

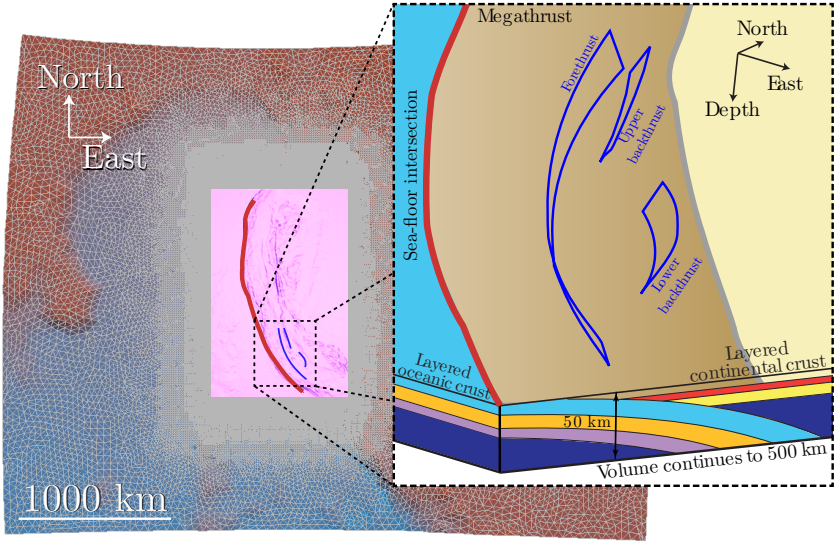


Figure 32: Shown is the bird's-eye view on the unstructured tetrahedral mesh used in this study. The pinks box marks topography data with 30″ resolution. Red curve marks the megathrust trace and blue curves mark splay fault traces. The cutout shows the layered sub-surface structure, each layer having different wave velocities. Figure adapted from [158].

basic properties [138]. Dynamic rupture simulations may improve our understanding of the fundamental processes of earthquake faulting, and provide physics-based forecasts on ground motions and sea-floor displacement. However, such simulations are quite expensive in comparison to classic methods based on a kinematic rupture description and an elastic half-space (e.g. Okada [118]). We therefore investigate the feasibility of a dynamic rupture simulation of the Sumatra earthquake in Section 12.1.1. First results and the impact on earthquake-tsunami modelling are discussed in Section 12.1.2.

12.1.1 Feasibility of large-scale dynamic rupture simulations

In the discretisation with highest resolution, topography data with 30″ resolution is used [161], the fault is resolved with an average edge length of 400 m, and material layers in the vicinity of the subduction zone are

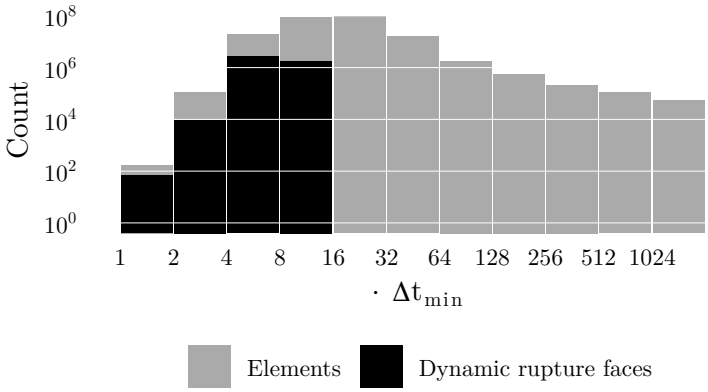


Figure 33: Time cluster histogram for elements and dynamic rupture faces. The time-step of almost all elements (98.7 %) lies in $[4\Delta t_{\min}, 64\Delta t_{\min})$ whereas the time-step of only 0.518 ‰ of elements is smaller than $4\Delta t_{\min}$. Figure from [158].

resolved with a 1.3–2.2 km resolution, such that about 2 Hz can be accurately resolved [82]. The mesh is coarsened away from the main area of study. The resulting unstructured mesh consists of 221 million elements.

The megathrust intersects the sea-floor at a very shallow angle and also intersects the material layers, as shown schematically in Figure 32. Sliver elements with insphere radii as low as 10 m are a consequence of the complicated geometry in combination with resolution requirements. (Even lower insphere radii, down to 6.1 cm, are observed using different meshing options.) The heterogeneous distribution of time-steps leads to a theoretical speed-up of 14.3, when perfect local time-stepping is employed. In a rate-2 clustered local time-stepping scheme, a theoretical speed-up of 9.9 is possible. Here, elements are distributed into 11 time clusters and dynamic rupture faces are distributed into 4 time clusters, cf. Figure 33.

Performance and scalability

We conducted a strong scaling experiment for the 221 million element mesh and the $\mathcal{O}6$ scheme [158]. The results for the baseline (BL) version [16, 64] and the Shaking Corals (SC) version are shown in Figure 34.

Global time-stepping. We observe a significant difference in global time-stepping performance between the BL version and the SC version. On

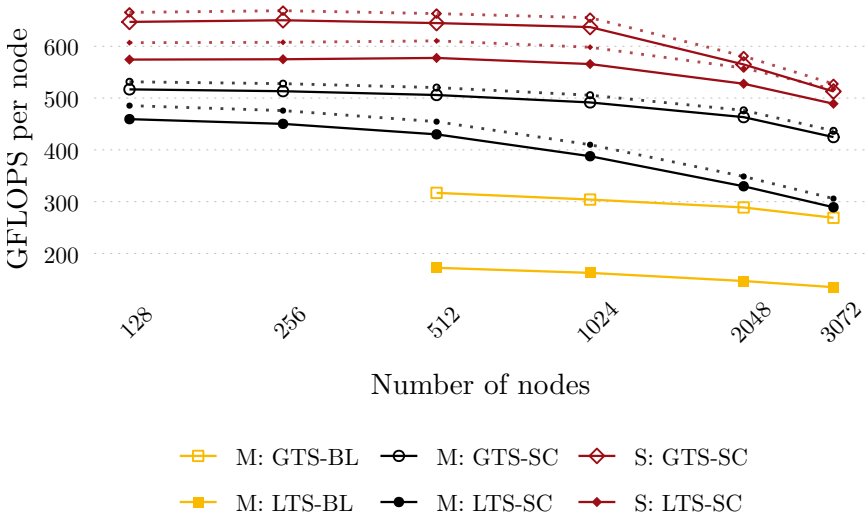


Figure 34: Strong scaling of the Sumatra earthquake using the 221 million element mesh. Figure compares the baseline (BL) version with the Shaking Corals (SC) version on SuperMUC Phase 2 (M) and Shaheen II (S). Solid lines only account for floating point operations in the wave propagation kernels, whereas dotted lines include floating point operations in the dynamic rupture kernels. Figure adapted from [158].

512 nodes of SuperMUC, performance increases by 189 GFLOP s^{-1} on average. The single node measurements in Section 11.2 show that the flux matrix decomposition used in the SC version gives a speed-up on SuperMUC but does not increase performance of the wave propagation kernels. Therefore, performance is for the most part "lost" in the dynamic rupture kernels of the BL version. While the BL version does not have a flop counter for dynamic rupture, an analysis of the source code shows that the most expensive part is the space time interpolation, which requires roughly $4 \cdot 10^6$ FLOP for $\mathcal{O}6$.[2] Strength reduction is not applied such that the number of flops is much higher than the optimal number of floating point operations. So even though the ratio of dynamic rupture faces to elements is 2 %, the dynamic rupture kernels can be roughly estimated to make up at least 10 % of the total floating point operations.

[2]Routine `Get_Extrapolated_Boundary_Values` in version `sc17_baseline`.

By considering the performance drop in comparison to the single node wave propagation performance, we estimate that the dynamic rupture kernels of the BL version run at about 6 % peak, which seems reasonable for non-optimised tensor contraction kernels. For the SC version, we infer from flop counters that about 5 % of the total flops are due to dynamic rupture. Moreover, GTS performance on 512 nodes closely matches the single node performance in Section 11.2.

On 3072 nodes, the GTS-SC version has a parallel efficiency of at least 79 % in comparison to 128 nodes.

Local time-stepping. The BL version does not allow local time-stepping with dynamic rupture. So in order test LTS for the BL version, we enforce that all elements adjacent to the fault have the same time-step. Here, the performance penalty of the non-optimised dynamic rupture kernels in the BL version is worse, because the relative update rate of dynamic rupture faces increases.

Parallel efficiency on 3072 nodes for LTS-SC is 63 % on SuperMUC Phase 2 and 85 % on Shaheen II in comparison to 128 nodes.

Load balance

The parallel efficiency in Figure 34 is worse than expected. For global time-stepping, over 85 % parallel efficiency is achieved in [16] on all 3072 nodes of SuperMUC Phase 2. While in the latter work a kinematic rupture simulation is tested, it is shown that a higher parallel efficiency should be possible.

In Section 8.4 we discussed that execution time of a test load is not necessarily homogeneous but a small percentage of compute nodes might be slower. We therefore employ run-time partitioning to weight each compute node by its execution time of a test load. Using this strategy on Shaheen II, we achieve a parallel efficiency of 98 % on 3072 nodes (in comparison to 128 nodes) and a parallel efficiency of 92 % on 4096 nodes for global time-stepping. For local-time stepping, we have 94 % on 3072 nodes and 91 % on 4096 nodes.

The opportunity to repeat the scaling run on SuperMUC Phase 2 did not arise, so we could not test if run-time partitioning is beneficial on this system. However, we had the opportunity to test strong scaling on Cori up to 6144 nodes. Here, we measured a parallel efficiency of 75 % for GTS and a parallel efficiency of 72 % for LTS on 6144 nodes.

An overview of the strong scaling results is shown in Figure 35. In these runs, additional data is obtained by newly developed internal monitoring tools: We place a stopwatch around each computational loop in

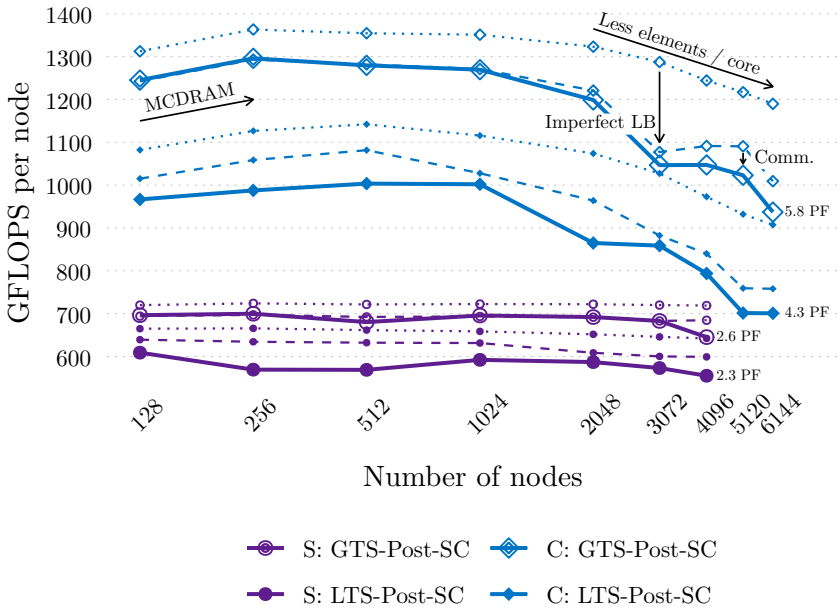


Figure 35: Strong scaling of the Sumatra earthquake on Shaheen II (S) and Cori (C). Run-time partitioning is used to account for execution time heterogeneity. Three metrics are shown: Performance based on average time spent in compute kernels (dotted), performance based on maximum time spent in compute kernels (dashed), and actual performance based on wall time (solid).

order to measure the time solely spent in computation. From this data, we can derive the actual load imbalance, which is determined by the ratio of average time spent in compute kernels to the maximum time spent in compute kernels. Moreover, the difference between the wall time and the maximum time spent in compute kernels can be attributed to communication overhead and the serial part of the application. Major effects are annotated in Figure 35.

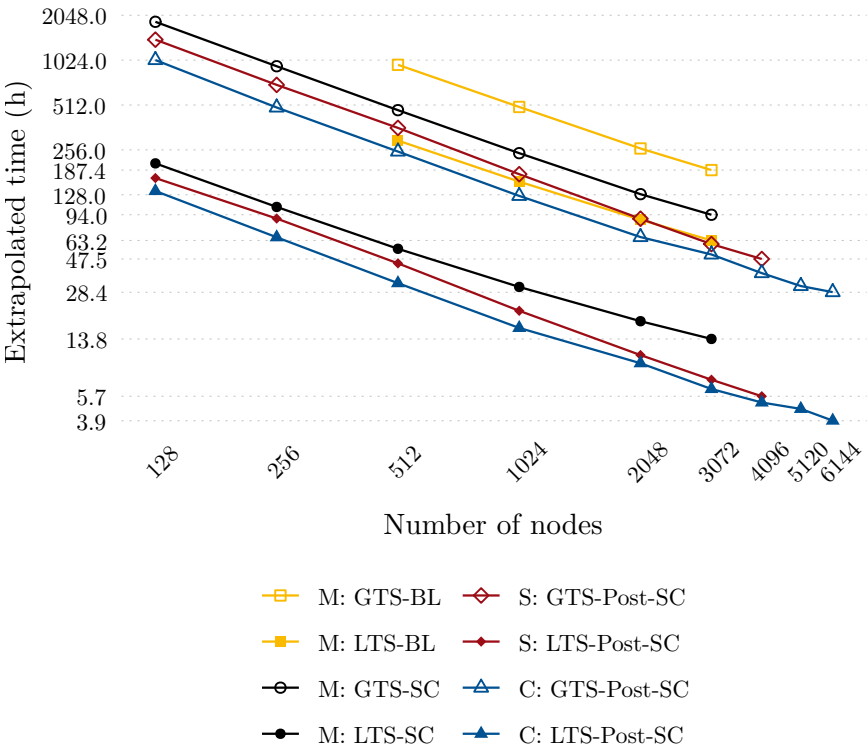


Figure 36: Extrapolated time-to-solution estimated from strong scaling runs. Shown is the baseline version and the Shaking Corals version, on SuperMUC Phase 2, Shaheen II, and Cori.

Time-to-solution

The wall time in the strong scaling runs is extrapolated to estimate the wall time for a full simulation with a duration of 500 s. An overview is shown in Figure 36. The baseline version is estimated to require more than 7 d to complete on 3072 nodes. Our ad-hoc implementation of LTS in the baseline version would require 2 d and 15.2 h, yielding a speed-up of 3.0 from the theoretical speed-up of 5.7. The GTS-SC version achieves a speed-up of 2.0 for GTS in comparison to the GTS-BL version, which is larger than the increase in performance, because less floating point operations have to be computed. Combined with local time-stepping, the SC version requires 13.8 h and achieves a speed-up of 6.8 in comparison

to GTS-SC (out of a theoretical maximum of 9.9) and a speed-up of 13.6 compared to GTS-BL. Given sufficient computational resources, the computation time can be reduced to 3.9 h on 6144 nodes of Cori.

Figure 36 only shows the extrapolated wall time. However, a production run on SuperMUC Phase 2 took 13.9 h, including 13 TB of checkpoint data and 2.8 TB of simulation output [158]. Thus we infer that one may obtain a realistic prediction of wall time from strong scaling results.

Conclusion

We now return to the core question whether large-scale dynamic rupture simulations of the 2004 Sumatra-Andaman earthquake are feasible. In principle, one could have run the baseline version. However, due to the typical time limits of 2–3 d on supercomputers, one would need to restart several times from checkpoints. Moreover, it is questionable whether one should invest about 16 million core hours (on SuperMUC Phase 2) for a single forward simulation. Our production run instead requires only 1.2 million core hours and completes within less than a day. Moreover, with optimised load balancing one would require only 0.75 million core hours on 4096 nodes of Shaheen II.

12.1.2 First results and outlook

Most of the basic earthquake properties of the production run [158] lie within the margin of uncertainty, which is inferred from observations [138]. The simulated earthquake has a moment magnitude of M_W 9.18 which lies in between the estimated M_W 9.1–9.3 . As shown in Figure 37, displacements match GPS observations [12] very well in orientation and magnitude. A large deviation is only visible close to the epicentre. The inset in Figure 37 also shows a significant contribution of splay faults to the vertical uplift. Rupture duration is 440 s in our model, which is faster than the rupture duration of 480–600 s obtained with source inversion methods [138].

The ability to run high-resolution dynamic rupture simulations allows the generation of various initial conditions for tsunami models, which enables studying the impact of source dynamics on tsunamigenesis. Initial conditions are for example the uplift due to the sea-floor displacement which may be plugged into the shallow water equations. The shallow water equations combined with a source term which accounts for bathymetry are commonly used for tsunami modelling.

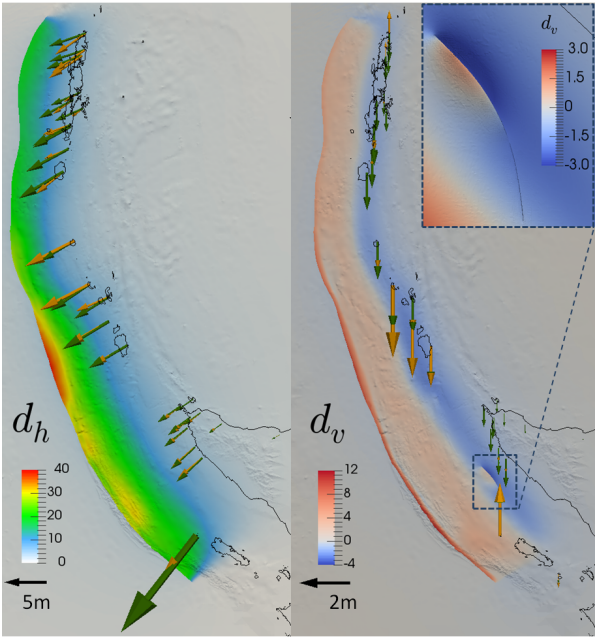


Figure 37: Comparison of synthetic displacements (green arrows) to GPS observations (orange arrows). The horizontal component is shown on the left and the vertical component on the right. Coloured bathymetry depicts the complete synthetic displacement field. Figure from [158].

In a pilot experiment, we quantify the impact of various uplift distributions on the Indian Ocean tsunami. The sea-floor displacements stem from dynamic rupture simulations with SeisSol by Wollherr [163], who compares the purely elastic model of the Sumatra-Andaman earthquake to a model which accounts for plastic yielding where stress exceeds the material's strength limit [164]. Uplift derived from the sea-floor displacement is plugged into the sam(oa)^2 software in which a Finite Volume scheme for the shallow water equations is implemented [111].

Uplift is prepared for samo(oa)^2 in two ways: The first method is to project the vertical component of the sea-floor displacement (e.g. right-hand figure in Figure 37) on a structured grid. The second method is to weight the horizontal components of the sea-floor displacement with the spatial derivatives of the bathymetry [149], add the resulting uplift to the

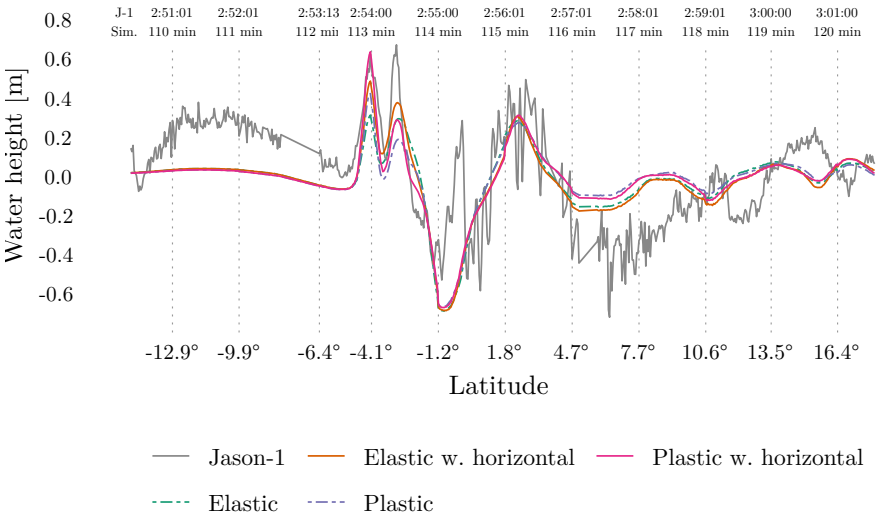


Figure 38: The Jason-1 altimetry satellite captured the Indian Ocean tsunami [58]. We compare the synthetic water height of four different models (see text) to the satellite's data. Water height is sampled minute-wise and combined within the figure in order to account for the flight duration of Jason-1.

vertical component of the sea-floor displacement, and project the uplift on a structured grid. In both cases, uplift is computed for a time-series in the interval of 0–500 s (with 0.5 s spacing) and applied gradually in the tsunami model. Besides the above-mentioned steps, no further filtering is applied.

When the horizontal components are neglected, uplift ranges from −4.1 m to 13.5 m for the elastic model and ranges from −4.4 m to 15.0 m for the plastic model. (Uplift is positive upwards.) When the horizontal components are considered, too, then uplift ranges from −4.2 m to 16.5 m for the elastic model and ranges from −5.2 m to 19.4 m for the plastic model.

The synthetic water heights obtained from the tsunami simulation is compared to satellite altimetry data in Figure 38. The largest differences between the models are seen at 4.1°W. Here, we see that the water height is ordered according to maximum uplift. In particular, the best fit is achieved for the plastic model which takes the horizontal components into account. The neighbouring peak towards 1.2°W is fit better by the elastic

model. Jason-1 is flying northwards with increasing latitude and at the same time the displacement of the plastic model is larger than the elastic model in the southern part of the fault and smaller than the elastic model in the northern part of the fault [163]. Therefore the spatial distribution of displacement might explain the alternating better fit of the plastic and the elastic model. For both peaks, accounting for the horizontal components of the sea-floor displacement leads to better matching water heights.

The pilot experiment demonstrates the importance of the rheological model but also the importance of the coupling details. In particular, accounting for horizontal displacement using the method of Tanioka et al. [149] has a large impact on the synthetic tsunami. Moreover, other authors advocate that one should use additional filtering to "smooth" the sea-floor displacement in space [56, 76, 116] and the question whether an initial horizontal velocity should be imposed is under debate [104].

A simple model extension would be to add a water layer on top of our Sumatra model. Assuming small perturbations, one may model the ocean with linear acoustics, which is a special case of the elastic wave equation (i.e. $\mu = 0$). An adjustment of the Riemann solver is necessary to properly couple elastic-acoustic interfaces as shown in [162] (see also Chapter 4). In addition, one needs to account for gravity in the water layer, e.g. by imposing a gravity boundary condition at the sea surface [103].

The computational cost of adding a water layer is likely minor (the model extends to 500 km and the maximum water depth is about 7 km), but could give valuable insights on the proper initial condition for tsunami models. Moreover, 3D coupled elastic-acoustic simulations could help understanding fast acoustic waves in the ocean which arrive before the tsunami and might be used to improve early warning systems [103].

12.2 The 2010 Darfield earthquake

The town of Darfield was hit by an M_W 7.1 earthquake on 3 September 2010 (16:35 UTC). Only about half a year later, the nearby city of Christchurch was hit by an M_W 6.3 earthquake on 21 February 2011 (23:51 UTC). The Christchurch earthquake inflicted huge structural damage and caused estimated losses of over NZ$ 20 billion [73].

For both events strong horizontal acceleration for periods less than 0.5 s was observed [73]. When these high frequency seismic waves shall be modelled in a physics-based earthquake simulation, one needs to account for viscoelastic attenuation, which strongly affects high frequency waves. Simulations with viscoelastic rheological models are more expensive than

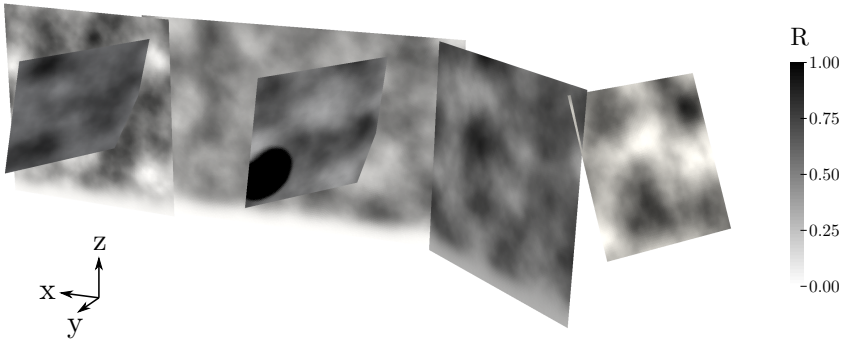


Figure 39: Fault geometry used in the Darfield model. Shown is the relative pre-stress ratio. The fault is overstressed within the black circle such that rupture nucleates therein.

purely elastic models, and, thinking of the hypothetical seismologist at the chapter's beginning, possibly billions of degrees of freedom are required to accurately resolve high frequency waves.

In order to test the feasibility of large-scale viscoelastic simulations, a model based on the Darfield earthquake is built, which is briefly discussed in Section 12.2.1. The largest mesh consists of 98,113,814 tetrahedra and requires 148 billion degrees of freedom for an $\mathcal{O}6$ simulation with three relaxation mechanisms. Performance and scalability on SuperMUC-NG, currently Europe's second largest supercomputer [112], is discussed in Section 12.2.2. We conclude with a brief discussion of the main features of the model and the impact of viscoelastic attenuation in Section 12.2.3.

12.2.1 Model description

The model consists of six fault planes proposed by Stramondo et al. [147]. As little is known about the exact stress field inside Earth, we follow the method of Ulrich et al. [153] and prescribe an initial stress field such that faulting on the six fault planes is favoured. The initial stress field is controlled by the relative pre-stress ratio R. We vary the latter randomly in order to create a heterogeneous initial stress field, see Figure 39.

The $15''$ topography and bathymetry grid from GEBCO [59] is used to generate a realistic free surface mesh. 3D velocity and Q models for New Zealand are employed [44, 45].

The CAD model, meshes, material, and fault parameterisation are available on Zenodo [154]. The dataset shows how easi (Chapter 9) is applied for this complex setup which includes data from various sources.

12.2.2 Performance and scalability

We test strong scalability for the 98 million element mesh and the viscoelastic rheological model (YATeTo version, split Kronecker variant). Figure 40 shows the results. Considering median values only, performance matches expectations from small-scale experiments in Section 11.5 and high parallel efficiency is obtained for GTS (over 93 %) and for LTS until 1536 nodes (over 84 %).

There are, however, quite significant outliers on SuperMUC-NG. In order to learn about their origin, we have plotted the load imbalance and the overhead. The latter are obtained by measuring the time spent in the three major compute kernels, i.e. the local update, neighbouring update, and friction law evaluation. For each rank we obtain the total time t_i spent in compute kernels. Relative load imbalance is defined as

$$L_R = 1 - \frac{\sum_{i=1}^{p} t_i}{p \cdot \max_i t_i} \tag{12.1}$$

and relative overhead is defined w.r.t. the wall time as

$$O_R = \frac{t_{\text{wall}}}{\max_i t_i} - 1. \tag{12.2}$$

It is striking that load imbalance seems to cluster around the median but some outliers have an absurdly high load imbalance. Let's recall the two implicit assumptions used in run-time partitioning:

1. Element weights passed to the partitioner represent the actual load.
2. Node weights obtained by measuring a unit load is representative throughout the whole run.

We cannot exclude that Assumption 1 is false but there seems to be no good reason why a pathological element weight distribution could lead to one node taking three times longer than the average node.

When rejecting Assumption 2 one wonders what could cause node performance variations. McCalpin [110] reports rarely occurring slow-downs of more than 30 % on Skylake Platinum. Slow-downs up to 20 % are reported for another implementation of ADER-DG [17], too. McCalpin

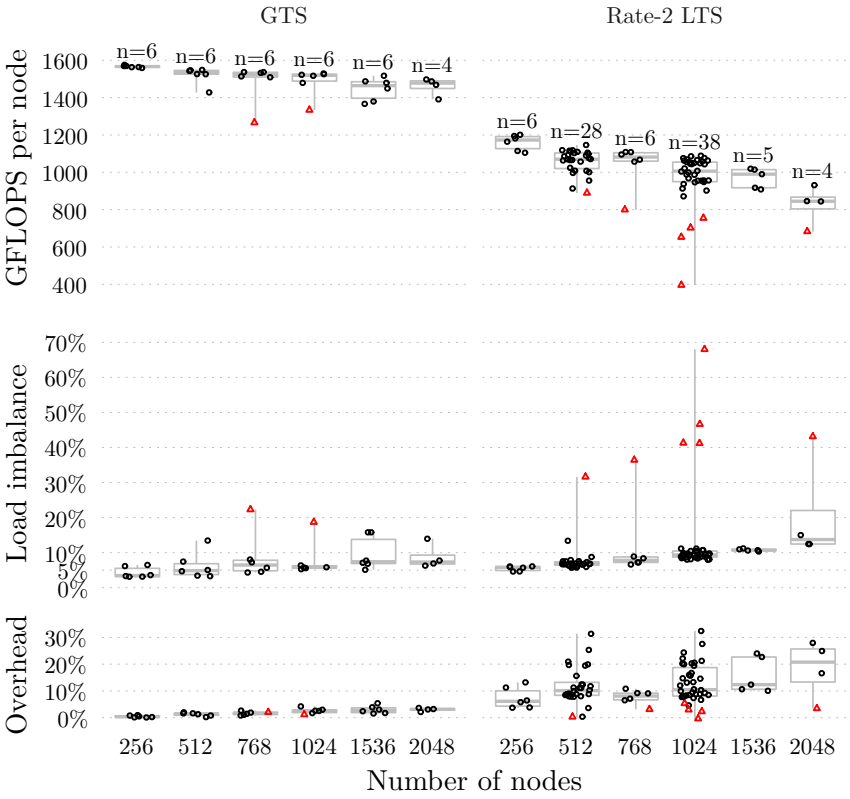
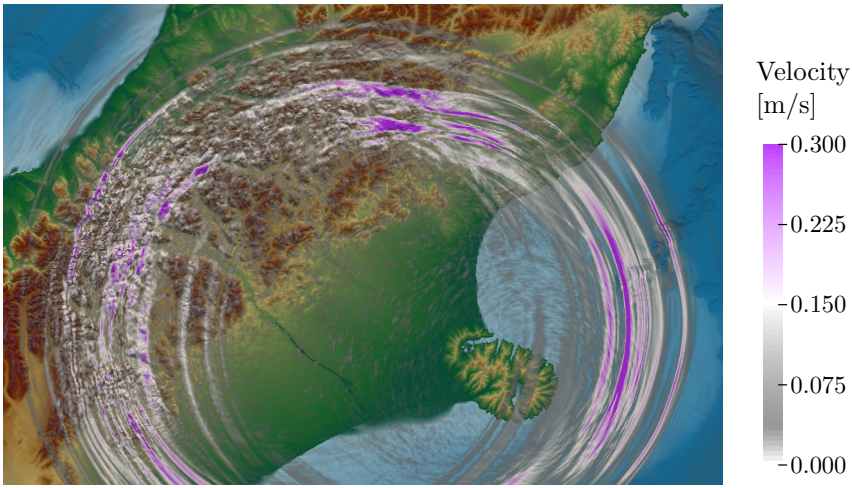


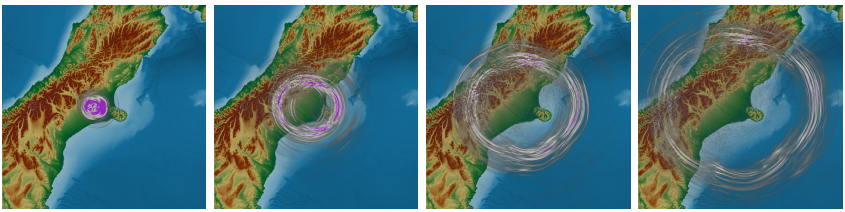
Figure 40: Strong scaling on SuperMUC-NG of the $\mathcal{O}6$ scheme with three relaxation mechanisms. Sample size n is given in the top plot. Large outliers in load imbalance are observed. (Runs with more than 16 % load imbalance are marked with red triangles.) Serial and communication overhead of LTS is much larger than for GTS.

[110] avoids slow-downs by using 1 GiB pages, whereas Breuer et al. [17] circumvent slow-downs with shared memory load-balancing.

The potential slow-downs are erratic and hard to track down. A total of 27 runs on 1024 nodes with LTS were run with enhanced monitoring output which might have given insight on the distribution of kernel execution times. However, an outlier appeared in none of these runs.



(a) $t = 40\,\mathrm{s}$



(b) $t = 10\,\mathrm{s}$ **(c)** $t = 30\,\mathrm{s}$ **(d)** $t = 50\,\mathrm{s}$ **(e)** $t = 70\,\mathrm{s}$

Figure 41: Snapshots of the particle velocity at the free surface.

When local time-stepping is enabled, large variability is observed in the overhead, too. Further research is necessary to clarify the source of this variability. However, the load-balancing issue is clearly more severe and needs to be addressed first.

12.2.3 Production run

A production run was computed in 2.87 h on 1536 nodes for a final simulation time of 75 s and a minimum time-step size of 63 µs. Local time-stepping was enabled. The simulation sustained $1.5\,\mathrm{PFLOP\,s^{-1}}$ with a load imbalance of 10.6 % and an overhead of 12.9 %, closely matching the median case in the strong scaling analysis.

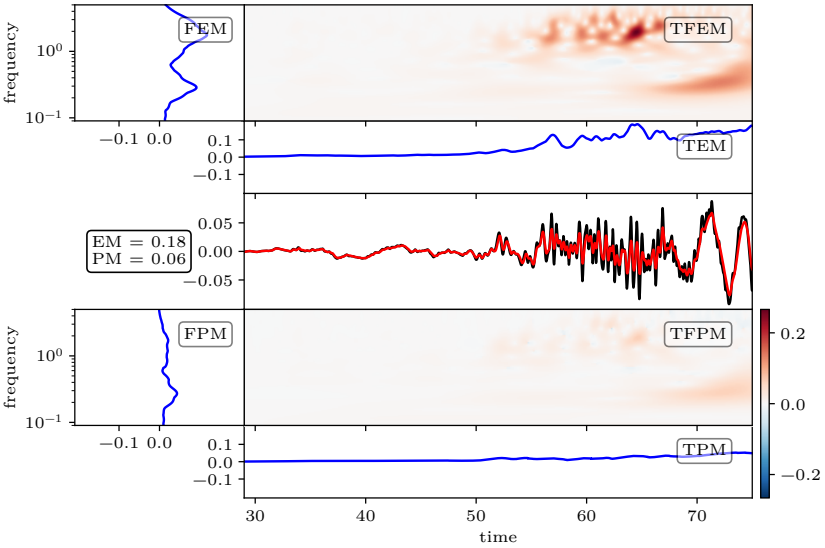


Figure 42: Comparison of vertical component of synthetic seismogram obtained with the elastic (black) and the viscoelastic (red) rheological model. Shown are envelope and phase misfit [92, 93]. The virtual seismograph is placed at 42.416 °S 173.539 °E close to the town of Kaikōura.

The simulated earthquake has a moment magnitude of M_W 7.16 which is slightly larger than the observed moment magnitude. Snapshots of the surface velocity are shown in Figure 41. The wave-field is strongly affected by the topography in the Southern Alps, as visible in Figure 41a, showing the importance of incorporating realistic topography data.

Simulating a viscoelastic rheological model is more expensive than an elastic rheological model. A production run of the latter completed in 1.24 h model, which makes the viscoelastic model 2.3× more expensive. The effect on particle velocity amplitudes is large, though, as shown for example in Figure 42.

CHAPTER

13

Conclusions

Earthquake models are becoming more and more complex and demand high efficiency and tremendous computing power. In this dissertation we addressed two seemingly conflicting goals: A sustainable code base which stands the test of time and highly efficient code which is tailored to the latest supercomputers. To this end, a DSL for small tensor operations is introduced and a compiler adapted to small tensor operations is developed.

Novel optimisation algorithms are developed. First, we introduce equivalent sparsity patterns and we prove that one may compute these using Boolean tensor operations. With equivalent sparsity patterns, essential optimisations in the ADER-DG scheme are found automatically, e.g. the vanishing derivatives in the Cauchy Kowalevski procedure. Moreover, one may dispense with a slice notation [86] altogether as sub-tensors involved in an operation are found automatically. Second, a dynamic programming algorithm is developed which automatically determines good index permutations of temporary tensors by minimising a heuristic cost function. Latter algorithm may also be used in the co-design of a numerical scheme and its software implementation, as it quickly shows how data layout choices map to Loop-over-GEMM.

The DSL is applied to implement elasticity, viscoelasticity, dynamic rupture, and plasticity [164] (not discussed in this thesis). Latter implementation allows to generate ADER-DG schemes for any order, any number of mechanisms, and any ensemble size from a single code base. The

support for general tensor operations enables a problem-adapted formulation for viscoelasticity which outperforms the matrix-based formulation. The previous implementation of elasticity is outperformed by introducing the flux matrix decomposition, which requires less floating point operations when strength reduction is applied and optimises cache usage. High performance is achieved for viscoelasticity, too, and results for ensemble simulations are competitive [20].

The DSL based on Einstein notation does not help in deriving the numerical scheme for a system of PDEs. It is, however, quite versatile and there is nothing which prevents its application to other algorithms which make heavy use of small tensor contractions. An interesting research direction would be the combination of the ideas and algorithms presented here with a high-level language for finite element discretisation, such as the Unified Form Language [4].

Besides addressing software complexity and optimisation, this thesis contributes to the field of physics-based rupture dynamics in multiple ways. First of all, semi-discrete stability of the ADER-DG scheme with frictional interface conditions is proven rigorously. Second, the inclusion of data from various sources for material and fault parameterisation is simplified with the easi library. While the benefit of easi is difficult to quantify, it evidently replaced thousands of lines of Fortran code and is used in the complicated setup of the 2016 Kaikōura earthquake [153]. Third, local time-stepping for dynamic rupture is implemented and load-balancing strategies are developed.

The extension of local-time stepping enabled the first dynamic rupture simulation of the 2004 Sumatra-Andaman earthquake. We conducted a production run with 221 million elements and up to 3.3 million time-steps on SuperMUC Phase 2. Here, a speed-up of 13.6 in comparison to the previous implementation and a speed-up of 6.8 compared to optimised global time-stepping is measured. Strong scaling experiments with improved load-balancing strategies revealed a parallel efficiency of 91 % on 4096 nodes of the Shaheen II supercomputer and a parallel efficiency of 72 % on 6144 nodes of the Cori supercomputer.

Some open problems are revealed that should be addressed in future work. First of all, alternative strategies for the neighbouring update should be explored, as for local time-stepping performance suffers from the low machine balance of SKX and potential future CPUs. Second, the rare and erratic load imbalances on SKX need to be addressed. Possible strategies could be huge pages [110] or dynamic shared memory load balancing [17]. Third, single precision arithmetic or mixed precision arith-

metic could potentially halve the execution time with sufficient accuracy for geophysical purposes.

Taken as a whole, we may now run large-scale kinematic and dynamic rupture simulations in a few hours on today's supercomputers. The workflow has reached a mature level and advanced rheological models and topography data can be rapidly included in a simulation. Hence, one should consider physics-based simulations for applications where simplified models predominate, e.g. in hazard assessment or source inversion.

Thank you for reading my thesis.

APPENDIX

A

Performance tables

Full benchmark results measured with the performance proxy application.
SMT indicates whether simultaneous multi-threading is used.
$\mathrm{NZ}_{\max}$ is the maximum observed non-zero performance in GFLOP s^{-1}.
$\mathrm{HW}_{\max}$ is the maximum observed hardware performance in GFLOP s^{-1}.
$\mathrm{HW}_{\mathrm{MAD}}$ is the maximum absolute deviation observed in HW.
Variants are explained in Chapter 11.

Table 14: Elastic wave equation. Source: [157].

$\mathcal{O}$	Variant	SMT	MDoF/s	$\mathrm{NZ}_{\max}$	$\mathrm{HW}_{\max}$	$\mathrm{HW}_{\mathrm{MAD}}$
KNL: Stampede 2						
Double precision						
2	sc.e.1	Yes	687	120	548	12.6
	yt.e.1	Yes	670	117	531	11.0
	yt.e.8	Yes	2262	392	443	15.3
	yt.e.16	Yes	1981	342	397	6.8
3	sc.e.1	Yes	1117	241	737	7.5
	yt.e.1	Yes	1115	240	732	13.7
	yt.e.8	No	2497	532	625	7.5
	yt.e.16	No	2295	488	573	29.5

Table 14: Elastic wave equation. Source: [157]. *(continued)*

$\mathcal{O}$	Variant	SMT	MDoF/s	NZ_{max}	HW_{max}	HW_{MAD}
4	sc.e.1	Yes	1211	341	1154	14.1
	yt.e.1	Yes	1184	334	1126	5.7
	yt.e.8	No	2122	598	742	1.9
	yt.e.16	Yes	1842	518	637	18.6
5	sc.e.1	Yes	1169	438	1314	21.4
	yt.e.1	Yes	1171	438	1314	14.5
	yt.e.8	Yes	1579	598	775	16.4
	yt.e.16	No	1370	518	664	16.8
6	sc.e.1	Yes	917	461	1435	6.9
	yt.e.1	Yes	912	459	1426	9.0
	yt.e.8	No	1134	581	792	13.2
	yt.e.16	Yes	876	447	601	20.9
7	sc.e.1	No	614	411	1343	79.9
	yt.e.1	No	605	405	1323	112.0
	yt.e.8	Yes	623	417	546	10.4
	yt.e.16	Yes	673	459	645	11.5
Single precision						
2	sc.e.1	Yes	770	134	1229	8.4
	yt.e.1	Yes	739	129	1172	36.8
	yt.e.16	Yes	3037	525	611	8.5
	yt.e.32	Yes	4034	697	808	12.1
3	sc.e.1	Yes	1502	324	1709	5.4
	yt.e.1	Yes	1427	307	1615	22.6
	yt.e.16	Yes	4882	1038	1224	12.3
	yt.e.32	No	3857	820	962	10.4
4	sc.e.1	Yes	1902	536	2122	56.1
	yt.e.1	Yes	1897	534	2108	83.2
	yt.e.16	No	4061	1143	1403	33.5
	yt.e.32	No	2961	833	1017	22.2
5	sc.e.1	Yes	1930	723	2549	55.8
	yt.e.1	Yes	1949	730	2568	55.6
	yt.e.16	Yes	3030	1145	1469	33.8
	yt.e.32	No	2163	817	1043	89.7
6	sc.e.1	Yes	1494	752	2902	5.6
	yt.e.1	Yes	1489	749	2889	1.4
	yt.e.16	No	2221	1135	1534	66.7

Table 14: Elastic wave equation. Source: [157]. *(continued)*

$\mathcal{O}$	Variant	SMT	MDoF/s	NZ_{max}	HW_{max}	HW_{MAD}
	yt.e.32	Yes	1603	818	1095	13.8
7	sc.e.1	Yes	1204	806	2888	10.4
	yt.e.1	Yes	1184	792	2839	11.2
	yt.e.16	Yes	1191	814	1142	10.0
	yt.e.32	Yes	1285	877	1220	17.6
SKX: SuperMUC-NG						
Double precision						
2	sc.e.1	Yes	414	72	330	0.7
	yt.e.1	Yes	427	74	339	0.2
	yt.e.8	Yes	1402	243	274	0.2
	yt.e.16	Yes	1609	278	318	0.9
3	sc.e.1	Yes	696	150	459	2.1
	yt.e.1	Yes	696	150	457	1.2
	yt.e.8	Yes	1659	353	415	0.6
	yt.e.16	Yes	1788	380	462	0.6
4	sc.e.1	Yes	1043	294	994	10.3
	yt.e.1	Yes	1043	294	992	3.9
	yt.e.8	Yes	1770	499	619	0.8
	yt.e.16	No	1827	514	669	0.2
5	sc.e.1	Yes	1212	454	1363	5.9
	yt.e.1	Yes	1199	449	1347	17.3
	yt.e.8	No	1807	684	887	2.9
	yt.e.16	No	1835	694	963	0.8
6	sc.e.1	Yes	1259	633	1970	28.6
	yt.e.1	Yes	1244	626	1946	42.5
	yt.e.8	No	1784	914	1246	3.1
	yt.e.16	No	1766	902	1332	4.0
7	sc.e.1	Yes	1048	702	2293	101.3
	yt.e.1	Yes	1024	686	2241	35.7
	yt.e.8	No	1707	1142	1495	27.0
	yt.e.16	No	1614	1103	1722	23.9
Single precision						
2	sc.e.1	Yes	510	89	815	3.8
	yt.e.1	Yes	530	92	840	0.3
	yt.e.16	Yes	2939	508	584	0.9

Table 14: Elastic wave equation. Source: [157]. *(continued)*

$\mathcal{O}$	Variant	SMT	MDoF/s	NZ_{max}	HW_{max}	HW_{MAD}
	yt.e.32	Yes	3300	570	653	3.9
3	sc.e.1	Yes	1249	269	1421	3.5
	yt.e.1	Yes	1344	290	1521	10.5
	yt.e.16	Yes	3377	718	877	3.2
	yt.e.32	Yes	3619	769	935	1.6
4	sc.e.1	Yes	1562	440	1741	6.0
	yt.e.1	Yes	1542	434	1714	26.8
	yt.e.16	Yes	3565	1003	1306	2.2
	yt.e.32	No	3687	1037	1343	0.5
5	sc.e.1	Yes	1901	712	2511	21.0
	yt.e.1	Yes	1877	703	2474	12.4
	yt.e.16	Yes	3655	1382	1918	1.5
	yt.e.32	Yes	3712	1402	1937	20.2
6	sc.e.1	No	1965	988	3816	21.7
	yt.e.1	No	1982	997	3845	49.8
	yt.e.16	Yes	3600	1840	2729	44.5
	yt.e.32	Yes	3544	1809	2660	58.4
7	sc.e.1	No	1827	1222	4383	36.0
	yt.e.1	No	1802	1206	4321	42.8
	yt.e.16	Yes	3275	2237	3493	68.0
	yt.e.32	No	3206	2187	3389	8.5

Table 15: Viscoelastic wave equation.

$\mathcal{O}$	Variant	SMT	MDoF/s	NZ_{max}	HW_{max}	HW_{MAD}
Double precision						
3 mechanisms						
2	yt.mc.1	Yes	506	82	262	0.3
	sc.uk.1	Yes	631	68	283	4.3
	yt.sk.1	Yes	760	94	285	1.2
	yt.sk.8	Yes	2440	301	330	2.9
	yt.sk.16	Yes	2702	333	369	1.0
3	yt.mc.1	No	1016	217	457	0.5
	sc.uk.1	Yes	1151	154	439	1.4
	yt.sk.1	Yes	1331	212	465	2.1

Table 15: Viscoelastic wave equation. *(continued)*

$\mathcal{O}$	Variant	SMT	MDoF/s	NZ_{max}	HW_{max}	HW_{MAD}
	yt.sk.8	Yes	2704	431	485	0.5
	yt.sk.16	Yes	2804	447	515	0.0
4	yt.mc.1	Yes	1603	405	843	4.5
	sc.uk.1	Yes	1692	299	946	0.0
	yt.sk.1	Yes	1906	407	964	6.0
	yt.sk.8	Yes	2572	553	638	0.1
	yt.sk.16	Yes	2600	558	673	0.3
5	yt.mc.1	Yes	1342	419	896	0.6
	sc.uk.1	Yes	1725	418	1212	0.8
	yt.sk.1	Yes	1788	554	1154	2.9
	yt.sk.8	Yes	2279	710	791	0.3
	yt.sk.16	Yes	2151	670	787	3.2
6	yt.mc.1	Yes	1309	521	1254	2.0
	sc.uk.1	Yes	1574	535	1597	2.4
	yt.sk.1	Yes	1604	664	1516	1.2
	yt.sk.8	Yes	1949	813	910	4.7
	yt.sk.16	No	1767	737	884	7.4
7	yt.mc.1	Yes	1053	546	1460	10.6
	sc.uk.1	Yes	1200	568	1749	38.6
	yt.sk.1	Yes	1214	675	1681	2.8
	yt.sk.8	No	1632	908	970	16.0
	yt.sk.16	No	1448	812	990	6.8
5 mechanisms						
2	yt.mc.1	Yes	591	94	294	0.6
	sc.uk.1	Yes	788	63	261	0.2
	yt.sk.1	Yes	969	97	278	1.7
	yt.sk.8	Yes	2984	299	324	0.4
	yt.sk.16	Yes	3283	329	359	0.6
3	yt.mc.1	Yes	1109	225	453	0.3
	sc.uk.1	Yes	1379	138	398	1.0
	yt.sk.1	Yes	1677	222	460	9.8
	yt.sk.8	Yes	3258	431	476	0.2
	yt.sk.16	Yes	3379	447	504	0.6
4	yt.mc.1	Yes	1727	401	757	19.4
	sc.uk.1	Yes	2012	267	828	1.5
	yt.sk.1	Yes	2316	409	889	2.9

Table 15: Viscoelastic wave equation. *(continued)*

	$\mathcal{O}$	Variant	SMT	MDoF/s	NZ_{max}	HW_{max}	HW_{MAD}
		yt.sk.8	Yes	3075	546	617	2.1
		yt.sk.16	Yes	3031	538	631	1.0
	5	yt.mc.1	Yes	1724	478	926	0.3
		sc.uk.1	Yes	2079	376	1071	1.8
		yt.sk.1	Yes	2183	547	1064	0.7
		yt.sk.8	Yes	2690	677	743	0.6
		yt.sk.16	No	2534	637	733	1.4
	6	yt.mc.1	Yes	1738	592	1277	1.6
		sc.uk.1	Yes	1997	501	1463	1.3
		yt.sk.1	Yes	1999	659	1394	4.0
		yt.sk.8	Yes	2288	759	838	1.3
		yt.sk.16	No	2074	688	808	6.0
	7	yt.mc.1	Yes	1379	591	1432	4.3
		sc.uk.1	Yes	1508	521	1578	20.0
		yt.sk.1	Yes	1523	663	1539	10.4
		yt.sk.8	No	1924	837	888	3.0
		yt.sk.16	No	1471	645	771	0.8
7 mechanisms							
	2	yt.mc.1	Yes	600	104	240	0.4
		sc.uk.1	Yes	914	60	246	0.4
		yt.sk.1	Yes	1127	99	270	5.4
		yt.sk.8	Yes	3380	297	319	0.3
		yt.sk.16	Yes	3681	323	349	0.8
	3	yt.mc.1	Yes	1158	229	449	3.7
		sc.uk.1	Yes	1538	128	369	0.5
		yt.sk.1	Yes	1888	223	444	7.8
		yt.sk.8	Yes	3655	431	469	0.7
		yt.sk.16	Yes	3750	442	490	0.4
	4	yt.mc.1	Yes	1807	400	708	0.6
		sc.uk.1	Yes	2212	242	737	0.6
		yt.sk.1	Yes	2612	411	834	4.4
		yt.sk.8	Yes	3429	542	602	2.7
		yt.sk.16	No	3337	527	605	0.9
	5	yt.mc.1	Yes	1764	456	825	1.3
		sc.uk.1	Yes	2301	341	958	0.4
		yt.sk.1	Yes	2463	540	994	1.7

Table 15: Viscoelastic wave equation. *(continued)*

	$\mathcal{O}$	Variant	SMT	MDoF/s	NZ_{max}	HW_{max}	HW_{MAD}
		yt.sk.8	No	2972	654	709	1.1
		yt.sk.16	No	2737	602	681	1.2
	6	yt.mc.1	Yes	1842	572	1135	5.3
		sc.uk.1	Yes	2318	472	1352	0.3
		yt.sk.1	Yes	2294	654	1299	1.2
		yt.sk.8	No	2495	715	781	1.0
		yt.sk.16	No	2164	620	716	4.9
	7	yt.mc.1	No	1493	568	1274	6.1
		sc.uk.1	Yes	1753	487	1452	8.6
		yt.sk.1	Yes	1740	646	1414	1.2
		yt.sk.8	No	2100	779	821	7.8
		yt.sk.16	No	1210	452	531	2.3
9 mechanisms							
	2	yt.mc.1	No	639	109	250	0.5
		sc.uk.1	Yes	1009	57	233	0.2
		yt.sk.1	Yes	1251	101	264	3.9
		yt.sk.8	Yes	3682	296	315	0.3
		yt.sk.16	Yes	3982	320	342	0.7
	3	yt.mc.1	Yes	1192	232	446	0.5
		sc.uk.1	Yes	1663	120	348	1.0
		yt.sk.1	Yes	2069	225	436	1.6
		yt.sk.8	Yes	3922	427	461	0.0
		yt.sk.16	Yes	3962	432	473	8.0
	4	yt.mc.1	Yes	1853	398	673	0.5
		sc.uk.1	Yes	2376	226	677	2.8
		yt.sk.1	Yes	2838	412	793	2.8
		yt.sk.8	Yes	3640	531	583	1.2
		yt.sk.16	No	3520	513	580	0.7
	5	yt.mc.1	Yes	1932	478	821	9.7
		sc.uk.1	Yes	2450	313	869	1.4
		yt.sk.1	Yes	2664	532	936	1.1
		yt.sk.8	No	3194	640	689	2.2
		yt.sk.16	No	2874	576	643	2.2
	6	yt.mc.1	Yes	2051	598	1114	5.4
		sc.uk.1	Yes	2559	446	1256	5.4
		yt.sk.1	Yes	2529	651	1227	3.2

Table 15: Viscoelastic wave equation. *(continued)*

$\mathcal{O}$	Variant	SMT	MDoF/s	NZ_{max}	HW_{max}	HW_{MAD}
	yt.sk.8	No	2638	683	739	2.5
	yt.sk.16	No	1982	513	583	4.3
7	yt.mc.1	Yes	1664	584	1230	3.7
	sc.uk.1	No	1981	468	1373	6.9
	yt.sk.1	Yes	1909	633	1318	11.7
	yt.sk.8	No	2174	721	756	5.0
	yt.sk.16	No	979	326	378	7.5
Single precision						
3 mechanisms						
2	yt.mc.1	Yes	625	101	645	8.3
	sc.uk.1	Yes	774	84	696	1.5
	yt.sk.1	Yes	943	117	700	7.1
	yt.sk.16	Yes	5118	631	700	2.0
	yt.sk.32	Yes	5574	687	761	1.4
3	yt.mc.1	Yes	1564	334	887	0.6
	sc.uk.1	Yes	1928	257	1167	0.4
	yt.sk.1	Yes	2183	348	1132	6.7
	yt.sk.16	Yes	5539	883	1021	0.3
	yt.sk.32	Yes	5671	904	1042	1.9
4	yt.mc.1	Yes	2404	607	1499	53.7
	sc.uk.1	Yes	2587	457	1633	5.9
	yt.sk.1	Yes	2903	620	1674	87.8
	yt.sk.16	Yes	5119	1099	1325	0.0
	yt.sk.32	Yes	5211	1119	1345	1.0
5	yt.mc.1	Yes	2937	916	2372	21.2
	sc.uk.1	Yes	3029	734	2600	6.9
	yt.sk.1	Yes	3110	964	2420	32.1
	yt.sk.16	Yes	4506	1403	1648	1.2
	yt.sk.32	Yes	4333	1348	1580	5.5
6	yt.mc.1	Yes	2986	1188	3424	7.5
	sc.uk.1	Yes	3065	1042	3729	3.4
	yt.sk.1	Yes	3085	1278	3488	18.9
	yt.sk.16	Yes	3862	1610	1937	15.3
	yt.sk.32	No	3556	1481	1774	7.0
7	yt.mc.1	Yes	2047	1061	3134	1.5

Table 15: Viscoelastic wave equation. *(continued)*

$\mathcal{O}$	Variant	SMT	MDoF/s	NZ_{max}	HW_{max}	HW_{MAD}
	sc.uk.1	Yes	2421	1146	3904	1.5
	yt.sk.1	Yes	2409	1339	3677	15.3
	yt.sk.16	No	3149	1765	2154	13.7
	yt.sk.32	No	2873	1610	1956	11.3
5 mechanisms						
2	yt.mc.1	Yes	752	120	747	0.1
	sc.uk.1	Yes	946	76	628	4.7
	yt.sk.1	Yes	1172	118	664	20.9
	yt.sk.16	Yes	6353	636	696	6.5
	yt.sk.32	Yes	6722	673	735	2.0
3	yt.mc.1	Yes	1859	378	928	4.5
	sc.uk.1	Yes	2362	237	1048	0.1
	yt.sk.1	Yes	2892	383	1134	6.3
	yt.sk.16	Yes	6664	881	996	0.4
	yt.sk.32	Yes	6845	905	1021	2.2
4	yt.mc.1	Yes	2659	618	1403	10.5
	sc.uk.1	Yes	3044	404	1430	6.0
	yt.sk.1	Yes	3457	611	1531	3.1
	yt.sk.16	Yes	6169	1096	1284	0.7
	yt.sk.32	Yes	6092	1082	1265	1.5
5	yt.mc.1	Yes	3324	921	2158	37.7
	sc.uk.1	Yes	3577	646	2249	8.4
	yt.sk.1	Yes	3832	961	2247	6.5
	yt.sk.16	Yes	5355	1347	1549	2.4
	yt.sk.32	No	5043	1268	1455	2.5
6	yt.mc.1	Yes	3485	1188	3051	3.6
	sc.uk.1	Yes	3819	958	3349	1.2
	yt.sk.1	Yes	3863	1274	3211	2.1
	yt.sk.16	Yes	4535	1504	1770	4.0
	yt.sk.32	No	4130	1369	1604	9.6
7	yt.mc.1	Yes	2702	1157	3094	0.8
	sc.uk.1	Yes	3091	1069	3575	7.8
	yt.sk.1	Yes	3029	1318	3371	7.8
	yt.sk.16	No	3713	1627	1945	16.7
	yt.sk.32	No	2968	1300	1548	11.6

Table 15: Viscoelastic wave equation. *(continued)*

$\mathcal{O}$	Variant	SMT	MDoF/s	NZ_{max}	HW_{max}	HW_{MAD}
7 mechanisms						
2	yt.mc.1	Yes	770	133	613	0.6
	sc.uk.1	Yes	1080	71	582	0.8
	yt.sk.1	Yes	1363	120	644	11.3
	yt.sk.16	Yes	7158	629	680	5.5
	yt.sk.32	Yes	7558	664	717	1.4
3	yt.mc.1	Yes	1931	382	894	9.7
	sc.uk.1	Yes	2691	223	964	2.6
	yt.sk.1	Yes	3306	390	1074	61.7
	yt.sk.16	Yes	7449	878	976	1.2
	yt.sk.32	Yes	7576	893	990	0.3
4	yt.mc.1	Yes	2783	617	1327	2.7
	sc.uk.1	Yes	3327	365	1277	3.5
	yt.sk.1	Yes	3982	626	1481	16.6
	yt.sk.16	Yes	6866	1084	1245	0.5
	yt.sk.32	No	6671	1053	1207	2.3
5	yt.mc.1	Yes	3509	908	1984	5.3
	sc.uk.1	Yes	3974	588	2016	1.4
	yt.sk.1	Yes	4296	942	2082	14.0
	yt.sk.16	Yes	5852	1287	1455	2.2
	yt.sk.32	No	5503	1210	1366	2.2
6	yt.mc.1	Yes	3740	1161	2738	1.7
	sc.uk.1	Yes	4322	880	3012	2.8
	yt.sk.1	Yes	4390	1252	2952	5.9
	yt.sk.16	No	4893	1402	1621	12.1
	yt.sk.32	No	4339	1243	1432	7.1
7	yt.mc.1	Yes	2924	1113	2751	108.6
	sc.uk.1	Yes	3621	1007	3313	6.1
	yt.sk.1	Yes	3491	1296	3122	2.1
	yt.sk.16	No	4093	1529	1796	15.7
	yt.sk.32	No	2525	943	1104	20.7
9 mechanisms						
2	yt.mc.1	Yes	838	143	654	1.1
	sc.uk.1	Yes	1177	67	545	1.6
	yt.sk.1	Yes	1473	119	611	7.9

Table 15: Viscoelastic wave equation. *(continued)*

$\mathcal{O}$	Variant	SMT	MDoF/s	NZ_{max}	HW_{max}	HW_{MAD}
	yt.sk.16	Yes	7767	623	669	3.9
	yt.sk.32	Yes	8136	653	699	0.8
3	yt.mc.1	Yes	2081	405	917	2.6
	sc.uk.1	Yes	2933	211	895	2.9
	yt.sk.1	Yes	3558	388	1008	16.6
	yt.sk.16	Yes	8021	874	959	1.1
	yt.sk.32	Yes	8033	875	959	1.2
4	yt.mc.1	Yes	2879	619	1282	3.3
	sc.uk.1	Yes	3566	339	1180	3.6
	yt.sk.1	Yes	4308	625	1414	60.3
	yt.sk.16	Yes	7292	1063	1201	7.2
	yt.sk.32	No	7039	1026	1157	2.1
5	yt.mc.1	Yes	3608	893	1853	16.0
	sc.uk.1	Yes	4270	546	1845	1.9
	yt.sk.1	Yes	4677	934	1972	20.6
	yt.sk.16	No	6247	1251	1397	2.4
	yt.sk.32	No	5711	1143	1275	6.6
6	yt.mc.1	Yes	3875	1130	2492	7.3
	sc.uk.1	Yes	4735	825	2772	10.8
	yt.sk.1	Yes	4775	1230	2743	12.5
	yt.sk.16	No	5156	1333	1520	3.6
	yt.sk.32	No	4073	1053	1197	15.6
7	yt.mc.1	Yes	3280	1151	2672	10.4
	sc.uk.1	Yes	4047	956	3098	5.3
	yt.sk.1	Yes	3869	1282	2936	26.1
	yt.sk.16	No	4287	1429	1656	7.8
	yt.sk.32	No	1980	660	762	22.1

Bibliography

[1] Milton Abramowitz and Irene A. Stegun. *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*. Tenth printing with corrections (1972). Washington, DC: United States Department of Commerce, 1964.

[2] K. Åhlander. "Einstein summation for multidimensional arrays". In: *Computers and Mathematics with Applications* 44.8 (2002), pp. 1007–1017. DOI: 10.1016/S0898-1221(02)00210-9.

[3] Keiiti Aki and Paul G. Richards. *Quantitative Seismology*. 2nd edition. University Science Books, 2002. ISBN: 0-935702-96-2.

[4] Martin S. Alnæs, Anders Logg, Kristian B. Ølgaard, Marie E. Rognes, and Garth N. Wells. "Unified Form Language: A Domain-specific Language for Weak Formulations of Partial Differential Equations". In: *ACM Transactions on Mathematical Software* 40.2 (Mar. 2014), 9:1–9:37. DOI: 10.1145/2566630.

[5] Jason Ansel, Shoaib Kamil, Kalyan Veeramachaneni, Jonathan Ragan-Kelley, Jeffrey Bosboom, Una-May O'Reilly, and Saman Amarasinghe. "OpenTuner: An Extensible Framework for Program Autotuning". In: *International Conference on Parallel Architectures and Compilation Techniques*. Edmonton, Canada, Aug. 2014. DOI: 10.1145/2628071.2628092.

[6] H. L. Atkins and Chi-Wang Shu. *Quadrature-Free Implementation of Discontinuous Galerkin Method for Hyperbolic Equations*. ICASE Report 96-51. 1996.

[7] Harold L. Atkins and Chi-Wang Shu. "Quadrature-Free Implementation of Discontinuous Galerkin Method for Hyperbolic Equations". In: *AIAA Journal* 36.5 (1998), pp. 775–782. DOI: 10.2514/2.436.

[8] Michael Barall. “A grid-doubling finite-element technique for calculating dynamic three-dimensional spontaneous rupture on an earthquake fault”. In: *Geophysical Journal International* 178.2 (2009), pp. 845–859. DOI: 10.1111/j.1365-246X.2009.04190.x.

[9] G. Baumgartner et al. “Synthesis of High-Performance Parallel Programs for a Class of ab Initio Quantum Chemistry Models”. In: *Proceedings of the IEEE* 93.2 (Feb. 2005), pp. 276–292. DOI: 10.1109/JPROC.2004.840311.

[10] Oren Ben-Kiki, Clark Evans, and Ingy döt Net. *YAML Ain't Markup Language*. https://yaml.org/spec/1.2/spec.html. Accessed: 2019-10-21. Oct. 2009.

[11] M. Benjemaa, N. Glinsky-Olivier, V. M. Cruz-Atienza, and J. Virieux. “3-D dynamic rupture simulations by a finite volume method”. In: *Geophysical Journal International* 178.1 (July 2009), pp. 541–560. DOI: 10.1111/j.1365-246X.2009.04088.x.

[12] Quentin Bletery, Anthony Sladen, Junle Jiang, and Mark Simons. “A Bayesian source model for the 2004 great Sumatra-Andaman earthquake”. In: *Journal of Geophysical Research: Solid Earth* 121.7 (2016), pp. 5116–5135. DOI: 10.1002/2016JB012911.

[13] Roger D. Borcherdt. “Energy and plane waves in linear viscoelastic media”. In: *Journal of Geophysical Research* 78.14 (1973), pp. 2442–2453. DOI: 10.1029/JB078i014p02442.

[14] Nathan W. Brei. “Generating Small Sparse Matrix Multiplication Kernels for Knights Landing”. MA thesis. Institut für Informatik, Technische Universität München, Feb. 2018.

[15] Susanne Brenner and Ridgway Scott. *The Mathematical Theory of Finite Element Methods*. 3rd ed. New York: Springer, 2008. ISBN: 978-0-387-75933-3.

[16] Alexander Nikolas Breuer. “High Performance Earthquake Simulations”. Dissertation. Technische Universität München, 2015. URL: https://nbn-resolving.org/urn:nbn:de:bvb:91-diss-20151221-1276756-1-4.

[17] Alexander Breuer, Yifeng Cui, and Alexander Heinecke. “Petaflop Seismic Simulations in the Public Cloud”. In: *ISC High Performance 2019*. Cham: Springer, 2019, pp. 167–185. ISBN: 978-3-030-20656-7.

[18] Alexander Breuer, Alexander Heinecke, and Michael Bader. “Petascale Local Time Stepping for the ADER-DG Finite Element Method”. In: *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. May 2016, pp. 854–863. DOI: 10.1109/IPDPS.2016.109.

[19] Alexander Breuer, Alexander Heinecke, Michael Bader, and Christian Pelties. “Accelerating SeisSol by Generating Vectorized Code for Sparse Matrix Operators”. In: *Parallel Computing: Accelerating Computational Science and Engineering (CSE)*. IOS Press, 2014, pp. 347–356. DOI: 10.3233/978-1-61499-381-0-347.

[20] Alexander Breuer, Alexander Heinecke, and Yifeng Cui. “EDGE: Extreme Scale Fused Seismic Simulations with the Discontinuous Galerkin Method”. In: *ISC High Performance 2017*. Cham: Springer, 2017, pp. 41–60. ISBN: 978-3-319-58667-0.

[21] Alexander Breuer, Alexander Heinecke, Leonhard Rannabauer, and Michael Bader. “High-Order ADER-DG Minimizes Energy- and Time-to-Solution of SeisSol”. In: *ISC High Performance 2015*. Cham: Springer, 2015, pp. 340–357. ISBN: 978-3-319-20119-1.

[22] Alexander Breuer, Alexander Heinecke, Sebastian Rettenberger, Michael Bader, Alice-Agnes Gabriel, and Christian Pelties. “Sustained Petascale Performance of Seismic Simulations with SeisSol on SuperMUC”. In: *ISC High Performance 2014*. Springer. 2014, pp. 1–18.

[23] José M. Carcione, Dan Kosloff, and Ronnie Kosloff. “Wave propagation simulation in a linear viscoelastic medium”. In: *Geophysical Journal International* 95.3 (1988), pp. 597–611. DOI: 10.1111/j.1365-246X.1988.tb06706.x.

[24] Leibniz Supercomputing Centre. *Hardware of SuperMUC-NG. Details of Compute Nodes*. https://doku.lrz.de/display/PUBLIC/Details+of+Compute+Nodes. Accessed: 2019-11-12. 2019.

[25] R. M. Christensen. *Theory of Viscoelasticity*. New York: Academic Press, 1982.

[26] Edith Cohen. “Structure Prediction and Computation of Sparse Matrix Products”. In: *Journal of Combinatorial Optimization* 2.4 (1998), pp. 307–332. DOI: 10.1023/A:1009716300509.

[27] Computational Infrastructure for Geodynamics. *SPECFEM3D Cartesian*. http://www.geodynamics.org/cig/software/specfem3d/. Accessed: 2019-11-06. 2019.

[28] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. 3rd ed. The MIT Press, 2009. ISBN: 978-0-262-03384-8.

[29] Intel Corporation. *Intel 64 and IA-32 Architectures Optimization Reference Manual*. Document no. 248966-033. Available at `https://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-optimization-manual.pdf`. June 2016.

[30] Intel Corporation. *Intel Xeon Phi Processor Software. Optimization Guide*. Document no. 334541-001. Available at `https://software.intel.com/sites/default/files/managed/11/56/intel-xeon-phi-processor-software-optimization-guide.pdf`. June 2016.

[31] Intel Corporation. *Intel Xeon Phi Processor: Your Path to Deeper Insight*. Available at `https://www.intel.com/content/dam/www/public/us/en/documents/product-briefs/xeon-phi-processor-product-brief.pdf`. 2016.

[32] Intel Corporation. *Intel Xeon Processor E5 v3 Product Family. Specification Update*. Document no. 330785-011. Available at `https://www.intel.com/content/dam/www/public/us/en/documents/specification-updates/xeon-e5-v3-spec-update.pdf`. Sept. 2017.

[33] Steven M. Day, Jacobo Bielak, Doug Dreger, Shawn Larsen, Robert Graves, Arben Pitarka, and Kim B. Olsen. *Tests of 3D elastodynamics codes: Final report for Lifelines program task 1A02*. Pacific Earthquake Engineering Research Center. Oct. 2003.

[34] Steven M. Day, Luis A. Dalguer, Nadia Lapusta, and Yi Liu. "Comparison of finite difference and boundary integral solutions to three-dimensional spontaneous rupture". In: *Journal of Geophysical Research* 110, B12307 (2005), pp. 1–23. DOI: 10.1029/2005JB003813.

[35] Edoardo Di Napoli, Diego Fabregat-Traver, Gregorio Quintana-Ortí, and Paolo Bientinesi. "Towards an efficient use of the BLAS library for multilinear tensor contractions". In: *Applied Mathematics and Computation* 235 (2014), pp. 454–468. DOI: 10.1016/j.amc.2014.02.051.

[36] Jack J. Dongarra, Jeremy Du Croz, Sven Hammarling, and Iain Duff. "A Set of Level 3 Basic Linear Algebra Subprograms". In: *ACM Transactions on Mathematical Software* 16.1 (Mar. 1990), pp. 1–17. DOI: 10.1145/77626.79170.

[37] Martin van Driel and Tarje Nissen-Meyer. “Optimized viscoelastic wave propagation for weakly dissipative media”. In: *Geophysical Journal International* 199.2 (Sept. 2014), pp. 1078–1093. DOI: 10.1093/gji/ggu314.

[38] Michael Dumbser. *Arbitrary High Order Schemes for the Solution of Hyperbolic Conservation Laws in Complex Domains*. Luft- und Raumfahrttechnik. Shaker Verlag, 2005. ISBN: 978-3-8322-4268-8.

[39] Michael Dumbser and Martin Käser. “An arbitrary high-order discontinuous Galerkin method for elastic waves on unstructured meshes – II. The three-dimensional isotropic case”. In: *Geophysical Journal International* 167 (2006), pp. 319–336. DOI: 10.1111/j.1365-246X.2006.03120.x.

[40] Michael Dumbser, Martin Käser, and Josep De La Puente. “Arbitrary high-order finite volume schemes for seismic wave propagation on unstructured meshes in 2D and 3D”. In: *Geophysical Journal International* 171.2 (Nov. 2007), pp. 665–694. DOI: 10.1111/j.1365-246X.2007.03421.x.

[41] Michael Dumbser, Martin Käser, and Eleuterio F. Toro. “An arbitrary high-order Discontinuous Galerkin method for elastic waves on unstructured meshes V: Local time stepping an p-adaptivity”. In: *Geophysical Journal International* 171.2 (2007), pp. 695–717. DOI: 10.1111/j.1365-246X.2007.03427.x.

[42] Kenneth Duru and Eric M. Dunham. “Dynamic earthquake rupture simulations on nonplanar faults embedded in 3D geometrically complex, heterogeneous elastic solids”. In: *Journal of Computational Physics* 305 (2016), pp. 185–207. DOI: 10.1016/j.jcp.2015.10.021.

[43] Adam M. Dziewonski and Don L. Anderson. “Preliminary reference Earth model”. In: *Physics of the Earth and Planetary Interiors* 25.4 (1981), pp. 297–356. DOI: 10.1016/0031-9201(81)90046-7.

[44] Donna Eberhart-Phillips, Stephen Bannister, and Martin Reyners. *New Zealand Wide model 2.1 seismic velocity model for New Zealand*. Zenodo. Version 2.1. Nov. 2017. DOI: 10.5281/zenodo.1043558.

[45] Donna Eberhart-Phillips and Bill Fry. *Data for: Joint local earthquake and teleseismic inversion for 3-D velocity and Q in New Zealand*. Mendely Data. Version 1.0. Aug. 2018. DOI: 10.17632/yy4f5frdm9.1.

[46] A. Einstein. “Die Grundlage der allgemeinen Relativitätstheorie”. In: *Annalen der Physik* 354.7 (1916), pp. 769–822. DOI: 10.1002/andp.19163540702.

[47] Helga Emmerich and Michael Korn. “Incorporation of attenuation into time-domain computations of seismic wave fields”. In: *Geophysics* 52.9 (Sept. 1987), pp. 1252–1264.

[48] Evgeny Epifanovsky et al. “New implementation of high-level correlated methods using a general block tensor library for high-performance electronic structure calculations”. In: *Journal of Computational Chemistry* 34.26 (2013), pp. 2293–2309. DOI: 10.1002/jcc.23377.

[49] V. Etienne, E. Chaljub, J. Virieux, and N. Glinsky. “An hp-adaptive discontinuous Galerkin finite-element method for 3-D elastic wave modelling”. In: *Geophysical Journal International* 183.2 (Nov. 2010), pp. 941–962. DOI: 10.1111/j.1365-246X.2010.04764.x.

[50] Manuel Fasching. “JIT compilation to realize flexible data access in simulation software”. Master’s thesis. Institut für Informatik, Technische Universität München, Mar. 2017. URL: http://www5.in.tum.de/pub/fasching_ma17.pdf.

[51] Andreas Fichtner and Martin van Driel. “Models and Fréchet kernels for frequency-(in)dependent Q”. In: *Geophysical Journal International* 198.3 (July 2014), pp. 1878–1889. DOI: 10.1093/gji/ggu228.

[52] Tiernan R. Fogarty and Randall J. LeVeque. “High-resolution finite-volume methods for acoustic waves in periodic and random media”. In: *The Journal of the Acoustical Society of America* 106.1 (1999), pp. 17–28. DOI: 10.1121/1.428038.

[53] Haohuan Fu et al. “18.9-Pflops Nonlinear Earthquake Simulation on Sunway TaihuLight: Enabling Depiction of 18-Hz and 8-meter Scenarios”. In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. SC ’17. Denver, Colorado: ACM, 2017, 2:1–2:12. ISBN: 978-1-4503-5114-0. DOI: 10.1145/3126908.3126910.

[54] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-oriented Software*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1995. ISBN: 0-201-63361-2.

[55] Gregor Gassner, Michael Dumbser, Florian Hindenlang, and Claus-Dieter Munz. "Explicit one-step time discretizations for discontinuous Galerkin and finite volume schemes based on local predictors". In: *Journal of Computational Physics* 230.11 (2011). Special issue High Order Methods for CFD Problems, pp. 4232–4247. DOI: 10.1016/j.jcp.2010.10.024.

[56] S. Glimsdal, G. K. Pedersen, C. B. Harbitz, and F. Løvholt. "Dispersion of tsunamis: does it really matter?" In: *Natural Hazards and Earth System Sciences* 13.6 (2013), pp. 1507–1526. DOI: 10.5194/nhess-13-1507-2013.

[57] Kazushige Goto and Robert A. van de Geijn. "Anatomy of high-performance matrix multiplication". In: *ACM Transactions on Mathematical Software* 34.3 (2008), 12:1–12:25. DOI: 10.1145/1356052.1356053.

[58] Jim Gower. "Jason 1 detects the 26 December 2004 tsunami". In: *Eos, Transactions American Geophysical Union* 86.4 (2005), pp. 37–38. DOI: 10.1029/2005EO040002.

[59] GEBCO Compilation Group. *GEBCO 2019 Grid*. 2019. DOI: 10.5285/836f016a-33be-6ddc-e053-6c86abc0788e.

[60] Georg Hager and Gerhard Wellein. *Introduction to High Performance Computing for Scientists and Engineers*. 1st ed. CRC Press, 2010. ISBN: 978-1-4398-1192-4.

[61] R. A. Harris et al. "The SCEC/USGS Dynamic Earthquake Rupture Code Verification Exercise". In: *Seismological Research Letters* 80.1 (2009), pp. 119–126. DOI: 10.1785/gssrl.80.1.119.

[62] Ruth A. Harris et al. "A Suite of Exercises for Verifying Dynamic Earthquake Rupture Codes". In: *Seismological Research Letters* 89.3 (2018), pp. 1146–1162.

[63] Albert Hartono et al. "Identifying Cost-Effective Common Subexpressions to Reduce Operation Count in Tensor Contraction Evaluations". In: *Computational Science – ICCS 2006*. Berlin, Heidelberg: Springer, 2006, pp. 267–275. ISBN: 978-3-540-34380-6.

[64] Alexander Heinecke, Alexander Breuer, Michael Bader, and Pradeep Dubey. "High Order Seismic Simulations on the Intel Xeon Phi Processor (Knights Landing)". In: *ISC High Performance 2016*. Springer, 2016, pp. 343–362. ISBN: 978-3-319-41321-1. DOI: 10.1007/978-3-319-41321-1_18.

[65] Alexander Heinecke, Alexander Breuer, and Yifeng Cui. "Tensor-optimized hardware accelerates fused discontinuous Galerkin simulations". In: *Parallel Computing* 89 (2019), p. 102550. DOI: 10.1016/j.parco.2019.102550.

[66] Alexander Heinecke, Greg Henry, Maxwell Hutchinson, and Hans Pabst. "LIBXSMM: Accelerating Small Matrix Multiplications by Runtime Code Generation". In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. SC '16. Salt Lake City, Utah: IEEE Press, 2016, 84:1–84:11. ISBN: 978-1-4673-8815-3. DOI: 10.1109/SC.2016.83.

[67] Alexander Heinecke et al. "Petascale High Order Dynamic Rupture Earthquake Simulations on Heterogeneous Supercomputers". In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. Gordon Bell Finalist. New Orleans: IEEE, Nov. 2014, pp. 3–14. ISBN: 9781479954995.

[68] Jan S. Hesthaven and Tim Warburton. *Nodal Discontinuous Galerkin Methods*. New York, USA: Springer, 2008. ISBN: 978-0-387-72065-4. DOI: 10.1007/978-0-387-72067-8.

[69] Maxwell Hutchinson, Alexander Heinecke, Hans Pabst, Greg Henry, Matteo Parsani, and David Keyes. "Efficiency of High Order Spectral Element Methods on Petascale Architectures". In: *ISC High Performance 2016*. Springer, 2016, pp. 449–466. ISBN: 978-3-319-41321-1. DOI: 10.1007/978-3-319-41321-1_23.

[70] Tsuyoshi Ichimura et al. "A Fast Scalable Implicit Solver for Nonlinear Time-evolution Earthquake City Problem on Low-ordered Unstructured Finite Elements with Artificial Intelligence and Transprecision Computing". In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis*. SC '18. Dallas, Texas: IEEE Press, 2018, 49:1–49:11. DOI: 10.1109/SC.2018.00052.

[71] Yoshiaki Ida. "Cohesive force across the tip of a longitudinal-shear crack and Griffith's specific surface energy". In: *Journal of Geophysical Research* 77.20 (1972), pp. 3796–3805. DOI: 10.1029/JB077i020p03796.

[72] W. Imperatori and P.M. Mai. "The role of topography and lateral velocity heterogeneities on near-source scattering and ground-motion variability". In: *Geophysical Journal International* 202.3 (July 2015), pp. 2163–2181. DOI: 10.1093/gji/ggv281.

[73] Earthquake Engineering Research Institute. *The M 6.3 Christchurch, New Zealand, Earthquake of February 22, 2011.* EERI Special Earthquake Report. May 2011, pp. 1–16.

[74] David D. Jackson and Don L. Anderson. "Physical mechanisms of seismic-wave attenuation". In: *Reviews of Geophysics* 8.1 (1970), pp. 1–63. DOI: 10.1029/RG008i001p00001.

[75] Jim Jeffers, James Reinders, and Avinash Sodani. "Vectorization with SDLT". In: *Intel Xeon Phi Processor High Performance Programming*. 2nd ed. Morgan Kaufmann, 2016. Chap. 11. ISBN: 978-0-12-809194-4.

[76] Kinjiro Kajiura. "The Leading Wave of a Tsunami". In: *Bulletin of the Earthquake Research Institute* 41 (1963), pp. 535–571.

[77] Y. Kaneko, N. Lapusta, and J.-P. Ampuero. "Spectral element modeling of spontaneous earthquake rupture on rate and state faults: Effect of velocity-strengthening friction at shallow depths". In: *Journal of Geophysical Research: Solid Earth* 113.B9 (2008). DOI: 10.1029/2007JB005553.

[78] G. Karypis and V. Kumar. "Multilevel Algorithms for Multi-Constraint Graph Partitioning". In: *SC '98: Proceedings of the 1998 ACM/IEEE Conference on Supercomputing*. Nov. 1998, pp. 28–28. DOI: 10.1109/SC.1998.10018.

[79] George Karypis and Kirk Schloegel. *ParMETIS. Parallel Graph Partitioning and Sparse Matrix Ordering Library.* http://glaros.dtc.umn.edu/gkhome/fetch/sw/parmetis/manual.pdf. Accessed: 2019-10-21. Mar. 2013.

[80] Martin Käser and Michael Dumbser. "A highly accurate discontinuous Galerkin method for complex interfaces between solids and moving fluids". In: *GEOPHYSICS* 73.3 (2008), T23–T35. DOI: 10.1190/1.2870081.

[81] Martin Käser, Michael Dumbser, Josep de la Puente, and Heiner Igel. "An arbitrary high-order Discontinuous Galerkin method for elastic waves on unstructured meshes – III. Viscoelastic attenuation". In: *Geophysical Journal International* 168 (2007), pp. 224–242. DOI: 10.1111/j.1365-246X.2006.03193.x.

[82] Martin Käser, Verena Hermann, and Josep de la Puente. "Quantitative accuracy analysis of the discontinuous Galerkin method for seismic wave propagation". In: *Geophysical Journal International* 173.3 (June 2008), pp. 990–999. DOI: 10.1111/j.1365-246X.2008.03781.x.

[83] Martin Käser, P. Martin Mai, and Michael Dumbser. "Accurate Calculation of Fault-Rupture Models Using the High-Order Discontinuous Galerkin Method on Tetrahedral Meshes". In: *Bulletin of the Seismological Society of America* 97.5 (2007), pp. 1570–1586.

[84] D. Kempf, R. Heß, S. Müthing, and P. Bastian. "Automatic Code Generation for High-Performance Discontinuous Galerkin Methods on Modern Architectures". In: *arXiv e-prints* (Dec. 2018). arXiv: arXiv:1812.08075 [math.NA].

[85] L. Knopoff. "Q". In: *Reviews of Geophysics* 2.4 (1964), pp. 625–660. DOI: 10.1029/RG002i004p00625.

[86] T. Kolda and B. Bader. "Tensor Decompositions and Applications". In: *SIAM Review* 51.3 (2009), pp. 455–500. DOI: 10.1137/07070111X.

[87] D. Komatitsch, S. Tsuboi, Chen Ji, and J. Tromp. "A 14.6 billion degrees of freedom, 5 teraflops, 2.5 terabyte earthquake simulation on the Earth Simulator". In: *SC '03: Proceedings of the 2003 ACM/IEEE Conference on Supercomputing*. Nov. 2003. DOI: 10.1145/1048935.1050155.

[88] Dimitri Komatitsch and Jeroen Tromp. "Introduction to the spectral element method for three-dimensional seismic wave propagation". In: *Geophysical Journal International* 139.3 (1999), pp. 806–822. DOI: 10.1046/j.1365-246x.1999.00967.x.

[89] B.V. Kostrov. "Selfsimilar problems of propagation of shear cracks". In: *Journal of Applied Mathematics and Mechanics* 28.5 (1964), pp. 1077–1087. DOI: https://doi.org/10.1016/0021-8928(64)90010-3.

[90] Jeremy E. Kozdon, Eric M. Dunham, and Jan Nordström. "Simulation of Dynamic Earthquake Ruptures in Complex Geometries Using High-Order Finite Difference Methods". In: *Journal of Scientific Computing* 55.1 (Apr. 2013), pp. 92–124. DOI: 10.1007/s10915-012-9624-5.

[91] Jozef Kristek and Peter Moczo. “Seismic-Wave Propagation in Viscoelastic Media with Material Discontinuities: A 3D Fourth-Order Staggered-Grid Finite-Difference Modeling”. In: *Bulletin of the Seismological Society of America* 93.5 (Oct. 2003), pp. 2273–2280. DOI: 10.1785/0120030023.

[92] Miriam Kristeková, Jozef Kristek, and Peter Moczo. “Time-frequency misfit and goodness-of-fit criteria for quantitative comparison of time signals”. In: *Geophysical Journal International* 178.2 (Aug. 2009), pp. 813–825. DOI: 10.1111/j.1365-246X.2009.04177.x.

[93] Miriam Kristeková, Jozef Kristek, Peter Moczo, and Steven M. Day. “Misfit Criteria for Quantitative Comparison of Seismograms”. In: *Bulletin of the Seismological Society of America* 96.5 (Oct. 2006), pp. 1836–1850. DOI: 10.1785/0120060012.

[94] Martin Kronbichler and Katharina Kormann. “Fast Matrix-Free Evaluation of Discontinuous Galerkin Finite Element Operators”. In: *ACM Transactions on Mathematical Software* 45.3 (Aug. 2019), 29:1–29:40. DOI: 10.1145/3325864.

[95] Chi Chung Lam. “Performance optimization of a class of loops implementing multi-dimensional integrals”. PhD thesis. UMI Company, 300 North Zeeb Road Ann Arbor, MI 48103: Graduate School of The Ohio State University, 1999. URL: http://rave.ohiolink.edu/etdc/view?acc_num=osu1488191667180786.

[96] Chi-Chung Lam, P. Sadayappan, Cociorva Daniel, Mebarek Alouani, and John Wilkins. “Performance Optimization of a Class of Loops Involving Sums of Products of Sparse Arrays”. In: *Ninth SIAM conference on Parallel Processing for Scientific Computing*. 1999.

[97] Chi-Chung Lam, P. Sadayappan, and Rephael Wenger. “Optimal reordering and mapping of a class of nested-loops for parallel execution”. In: *Languages and Compilers for Parallel Computing*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1997, pp. 315–329. ISBN: 978-3-540-69128-0.

[98] L Lambrecht, A Lamert, W Friederich, T Möller, and M S Boxberg. “A nodal discontinuous Galerkin approach to 3-D viscoelastic wave propagation in complex geological media”. In: *Geophysical Journal International* 212.3 (Nov. 2017), pp. 1570–1587. DOI: 10.1093/gji/ggx494.

[99] G. Laske, G. Masters, Z. Ma, and M. Pasyanos. "Update on CRUST1.0 - A 1-degree Global Model of Earth's Crust". In: *EGU General Assembly Conference Abstracts*. Vol. 15. EGU General Assembly Conference Abstracts. Apr. 2013. URL: http://igppweb.ucsd.edu/~gabi/crust1.html.

[100] Randall J. LeVeque. *Finite Volume Methods for Hyperbolic Problems*. Cambridge Texts in Applied Mathematics. Cambridge University Press, 2002. DOI: 10.1017/CBO9780511791253.

[101] J. Li, C. Battaglino, I. Perros, J. Sun, and R. Vuduc. "An input-adaptive and in-place approach to dense tensor-times-matrix multiply". In: *SC '15: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. Nov. 2015, pp. 1–12. DOI: 10.1145/2807591.2807671.

[102] Hsi-Ping Liu, Don L. Anderson, and Hiroo Kanamori. "Velocity dispersion due to anelasticity; implications for seismology and mantle composition". In: *Geophysical Journal International* 47.1 (1976), pp. 41–58. DOI: 10.1111/j.1365-246X.1976.tb01261.x.

[103] Gabriel C. Lotto and Eric M. Dunham. "High-order finite difference modeling of tsunami generation in a compressible ocean from offshore earthquakes". In: *Computational Geosciences* 19.2 (Apr. 2015), pp. 327–340. DOI: 10.1007/s10596-015-9472-0.

[104] Gabriel C. Lotto, Gabriel Nava, and Eric M. Dunham. "Should tsunami simulations include a nonzero initial horizontal velocity?" In: *Earth, Planets, and Space* 69.117 (2017), pp. 1–14. DOI: 10.1186/s40623-017-0701-8.

[105] Fabio Luporini, David A. Ham, and Paul H. J. Kelly. "An Algorithm for the Optimization of Finite Element Integration Loops". In: *ACM Transactions on Mathematical Software* 44.1 (Mar. 2017), 3:1–3:26. DOI: 10.1145/3054944.

[106] Fabio Luporini, Ana Lucia Varbanescu, Florian Rathgeber, Gheorghe-Teodor Bercea, J. Ramanujam, David A. Ham, and Paul H. J. Kelly. "Cross-Loop Optimization of Arithmetic Intensity for Finite Element Local Assembly". In: *ACM Transactions on Architecture and Code Optimization* 11.4 (Jan. 2015), 57:1–57:25. DOI: 10.1145/2687415.

[107] P. Martin Mai and K. K. S. Thingbaijam. "SRCMOD: An Online Database of Finite-Fault Rupture Models". In: *Seismological Research Letters* 85.6 (2015), pp. 1348–1357.

[108] P. Martin Mai et al. “The Earthquake-Source Inversion Validation (SIV) Project”. In: *Seismological Research Letters* 87.3 (2016), pp. 690–708.

[109] D. Matthews. “High-Performance Tensor Contraction without Transposition”. In: *SIAM Journal on Scientific Computing* 40.1 (2018), pp. C1–C24. DOI: 10.1137/16M108968X.

[110] John D. McCalpin. “HPL and DGEMM Performance Variability on the Xeon Platinum 8160 Processor”. In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis*. SC ’18. Dallas, Texas: IEEE Press, 2018, 18:1–18:13. DOI: 10.1109/SC.2018.00021.

[111] Oliver Meister, Kaveh Rahnema, and Michael Bader. “Parallel Memory-Efficient Adaptive Mesh Refinement on Structured Triangular Meshes with Billions of Grid Cells”. In: *ACM Transactions on Mathematical Software* 43.3 (Sept. 2016), 19:1–19:27. DOI: 10.1145/2947668.

[112] Hans Meuer, Erich Strohmaier, Jack Dongarra, Horst Simon, and Martin Meuer. *TOP500 List – November 2019*. https://www.top500.org/lists/2019/11/. Accessed: 2019-11-22. Nov. 2019.

[113] Peter Moczo and Jozef Kristek. “On the rheological models used for time-domain methods of seismic wave propagation”. In: *Geophysical Research Letters* 32.1 (Jan. 2005).

[114] T. Nelson, A. Rivera, P. Balaprakash, M. Hall, P. D. Hovland, E. Jessup, and B. Norris. “Generating Efficient Tensor Contractions for GPUs”. In: *2015 44th International Conference on Parallel Processing*. Sept. 2015, pp. 969–978. DOI: 10.1109/ICPP.2015.106.

[115] Tarje Nissen-Meyer, Martin van Driel, Simon Stähler, Kasra Hosseini, Stefanie Hempel, and Alexandre Fournier. *AxiSEM user manual v1.3*. https://geodynamics.org/cig/software/axisem/. Accessed: 2019-07-26. 2019.

[116] Mikhail A. Nosov and Sergey V. Kolesov. “Optimal Initial Conditions for Simulation of Seismotectonic Tsunamis”. In: *Pure and Applied Geophysics* 168.6 (June 2011), pp. 1223–1237. DOI: 10.1007/s00024-010-0226-6.

[117] R. J. O’Connell and B. Budiansky. “Measures of dissipation in viscoelastic media”. In: *Geophysical Research Letters* 5.1 (1978), pp. 5–8. DOI: 10.1029/GL005i001p00005.

[118] Yoshimitsu Okada. "Surface deformation due to shear and tensile faults in a half-space". In: *Bulletin of the Seismological Society of America* 75.4 (Aug. 1985), pp. 1135–1154. eprint: https://pubs.geoscienceworld.org/bssa/article-pdf/75/4/1135/2705188/BSSA0750041135.pdf.

[119] Kim Olsen, Steven Day, and Yifeng Cui. *AWP-ODC user manual*. http://hpgeoc.sdsc.edu/AWPODC/. Accessed: 2019-07-26. 2019.

[120] Elmar Peise and Paolo Bientinesi. "Performance Modeling for Dense Linear Algebra". In: *Proceedings of the 2012 SC Companion: High Performance Computing, Networking Storage and Analysis*. Washington, DC, USA: IEEE Computer Society, 2012, pp. 406–416. DOI: 10.1109/SC.Companion.2012.60.

[121] Elmar Peise, Diego Fabregat-Traver, and Paolo Bientinesi. "On the Performance Prediction of BLAS-based Tensor Contractions". In: *Proceedings of PMBS 2014*. Lecture Notes in Computer Science, vol. 8966. Springer, Apr. 2015, pp. 193–212. DOI: 10.1007/978-3-319-17248-4_10.

[122] C. Pelties, A.-A. Gabriel, and J.-P. Ampuero. "Verification of an ADER-DG method for complex dynamic rupture problems". In: *Geoscientific Model Development* 7.3 (2014), pp. 847–866. DOI: 10.5194/gmd-7-847-2014.

[123] Christian Pelties, Josep de la Puente, Jean-Paul Ampuero, Gilbert B. Brietzke, and Martin Käser. "Three-dimensional dynamic rupture simulation with a high-order discontinuous Galerkin method on unstructured tetrahedral meshes". In: *Journal of Geophysical Research: Solid Earth* 117.B2 (2012). DOI: 10.1029/2011JB008857.

[124] Josep de la Puente, J.-P. Ampuero, and Martin Käser. "Dynamic rupture modeling on unstructured meshes using a discontinuous Galerkin method". In: *Journal of Geophysical Research: Solid Earth* 114.B10 (2009). DOI: 10.1029/2008JB006271.

[125] Josep de la Puente, Michael Dumbser, Martin Käser, and Heiner Igel. "Discontinuous Galerkin methods for wave propagation in poroelastic media". In: *GEOPHYSICS* 73.5 (2008), T77–T97. DOI: 10.1190/1.2965027.

[126] Josep de la Puente, Martin Käser, Michael Dumbser, and Heiner Igel. "An arbitrary high-order Discontinuous Galerkin method for elastic waves on unstructured meshes – IV. Anisotropy". In: *Geophysical Journal International* 169 (2007), pp. 1210–1228. DOI: 10.1111/j.1365-246X.2007.03381.x.

[127] Florian Rathgeber et al. “Firedrake: Automating the Finite Element Method by Composing Abstractions”. In: *ACM Transactions on Mathematical Software* 43.3 (Dec. 2016), 24:1–24:27. DOI: 10.1145/2998441.

[128] Stefan Reinalter. *Implementing a semi-automatic structure-of-arrays data container*. https://blog.molecular-matters.com/2013/10/22/implementing-a-semi-automatic-structure-of-arrays-data-container/. Accessed: 2019-10-18. 2013.

[129] Sebastian Rettenberger. “Scalable I/O on Modern Supercomputers for Simulations on Unstructured Meshes”. Dissertation. Technische Universität München, 2018. ISBN: 978-3-8439-3586-9.

[130] Sebastian Rettenberger, Oliver Meister, Michael Bader, and Alice-Agnes Gabriel. “ASAGI - A Parallel Server for Adaptive Geoinformation”. In: *EASC '16 Proceedings of the Exascale Applications and Software Conference 2016*. ACM, Sept. 2016, 2:1–2:9. DOI: 10.1145/2938615.2938618.

[131] M. Rietmann et al. “Forward and adjoint simulations of seismic wave propagation on emerging large-scale GPU architectures”. In: *SC '12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. Nov. 2012, pp. 1–11. DOI: 10.1109/SC.2012.59.

[132] M. Rognes, R. Kirby, and A. Logg. “Efficient Assembly of $H(\mathrm{div})$ and $H(\mathrm{curl})$ Conforming Finite Elements”. In: *SIAM Journal on Scientific Computing* 31.6 (2010), pp. 4130–4151. DOI: 10.1137/08073901X.

[133] D. Roten, K. B. Olsen, S. M. Day, Y. Cui, and D. Fäh. “Expected seismic shaking in Los Angeles reduced by San Andreas fault zone plasticity”. In: *Geophysical Research Letters* (2014).

[134] Daniel Roten et al. “High-Frequency Nonlinear Earthquake Simulations on Petascale Heterogeneous Supercomputers”. In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE. 2016, 82:1–82:12.

[135] Kirk Schloegel, George Karypis, and Vipin Kumar. “Parallel static and dynamic multi-constraint graph partitioning”. In: *Concurrency and Computation: Practice and Experience* 14.3 (2002), pp. 219–240. DOI: 10.1002/cpe.605.

[136] Helmut Seidl, Reinhard Wilhelm, and Sebastian Hack. *Compiler Design: Analysis and Transformation*. Springer, 2012. ISBN: 978-3-642-17548-0.

[137] National Geophysical Data Center / World Data Service. *Global Historical Tsunami Database*. Accessed: 2019-11-29. 2019. DOI: 10.7289/V5PN93H7.

[138] Peter Shearer and Roland Bürgmann. "Lessons Learned from the 2004 Sumatra-Andaman Megathrust Rupture". In: *Annual Review of Earth and Planetary Sciences* 38 (2010), pp. 103–131. DOI: 10.1146/annurev-earth-040809-152537.

[139] Y. Shi, U. N. Niranjan, A. Anandkumar, and C. Cecka. "Tensor Contractions with Extended BLAS Kernels on CPU and GPU". In: *2016 IEEE 23rd International Conference on High Performance Computing (HiPC)*. Dec. 2016, pp. 193–202. DOI: 10.1109/HiPC.2016.031.

[140] Patrick Small et al. "The SCEC Unified Community Velocity Model Software Framework". In: *Seismological Research Letters* 88.6 (Sept. 2017), pp. 1539–1552. DOI: 10.1785/0220170082.

[141] J. S. Sochacki, J. H. George, R. E. Ewing, and S. B. Smithson. "Interface conditions for acoustic and elastic wave propagation". In: *GEOPHYSICS* 56.2 (1991), pp. 168–181. DOI: 10.1190/1.1443029.

[142] E. Solomonik, D. Matthews, J. Hammond, and J. Demmel. "Cyclops Tensor Framework: Reducing Communication and Eliminating Load Imbalance in Massively Parallel Contractions". In: *2013 IEEE 27th International Symposium on Parallel and Distributed Processing*. May 2013, pp. 813–824. DOI: 10.1109/IPDPS.2013.112.

[143] Edgar Solomonik, Devin Matthews, Jeff R. Hammond, John F. Stanton, and James Demmel. "A massively parallel tensor contraction framework for coupled-cluster computations". In: *Journal of Parallel and Distributed Computing* 74.12 (2014). Domain-Specific Languages and High-Level Frameworks for High-Performance Computing, pp. 3176–3190. DOI: https://doi.org/10.1016/j.jpdc.2014.06.002.

[144] Paul Springer and Paolo Bientinesi. "Design of a High-Performance GEMM-like Tensor-Tensor Multiplication". In: *ACM Transactions on Mathematical Software* 44.3 (2018), 28:1–28:29. DOI: 10.1145/3157733.

[145] S. Stein and M. Wysession. *An introduction to seismology, earthquakes and Earth structure*. Blackwell Publishing, 2003.

[146] Kevin Stock, Tom Henretty, Iyyappa Murugandi, P. Sadayappan, and Robert Harrison. "Model-Driven SIMD Code Generation for a Multi-Resolution Tensor Kernel". In: *Proceedings of the 2011 IEEE Parallel and Distributed Processing Symposium*. IEEE Computer Society, 2011, pp. 1058–1067. DOI: 10.1109/IPDPS.2011.101.

[147] Salvatore Stramondo et al. "Did the September 2010 (Darfield) earthquake trigger the February 2011 (Christchurch) event?" In: *Scientific Reports* 1 (2011), 98:1–98:7. DOI: 10.1038/srep00098.

[148] J. Tago, V. M. Cruz-Atienza, J. Virieux, V. Etienne, and F. J. Sánchez-Sesma. "A 3D hp-adaptive discontinuous Galerkin method for modeling earthquake dynamics". In: *Journal of Geophysical Research: Solid Earth* 117.B9 (2012). DOI: 10.1029/2012JB009313.

[149] Yuichiro Tanioka and Kenji Satake. "Tsunami generation by horizontal displacement of ocean bottom". In: *Geophysical Research Letters* 23.8 (1996), pp. 861–864. DOI: 10.1029/96GL00736.

[150] Maurizio Tavelli, Michael Dumbser, Dominic Etienne Charrier, Leonhard Rannabauer, Tobias Weinzierl, and Michael Bader. "A simple diffuse interface approach on adaptive Cartesian grids for the linear elastic wave equations with complex topography". In: *Journal of Computational Physics* 386 (2019), pp. 158–189. DOI: 10.1016/j.jcp.2019.02.004.

[151] Eleuterio F. Toro. *Riemann Solvers and Numerical Methods for Fluid Dynamics. A Practical Introduction*. Berlin, Heidelberg: Springer, 2009. ISBN: 978-3-540-49834-6.

[152] T. Ulrich et al. "Coupled, Physics-Based Modeling Reveals Earthquake Displacements are Critical to the 2018 Palu, Sulawesi Tsunami". In: *Pure and Applied Geophysics* 176.10 (Oct. 2019), pp. 4069–4109. DOI: 10.1007/s00024-019-02290-5.

[153] Thomas Ulrich, Alice-Agnes Gabriel, Jean-Paul Ampuero, and Wenbin Xu. "Dynamic viability of the 2016 Mw 7.8 Kaikōura earthquake cascade on weak crustal faults". In: *Nature Communications* 10 (2019), 1213:1–1213:16. DOI: 10.1038/s41467-019-09125-w.

[154] Carsten Uphoff. *Setup of the 2010 Darfield earthquake for SeisSol*. Zenodo. Dec. 2019. DOI: 10.5281/zenodo.3565774.

[155] Carsten Uphoff and Michael Bader. *EASI - A library for the easy setup of large scale earthquake simulations and other applications.* https://mediatum.ub.tum.de/1506266. Poster presented at deRSE19. Potsdam, June 2019.

[156] Carsten Uphoff and Michael Bader. "Generating high performance matrix kernels for earthquake simulations with viscoelastic attenuation". In: *2016 International Conference on High Performance Computing and Simulation (HPCS)*. July 2016, pp. 908–916. DOI: 10.1109/HPCSim.2016.7568431.

[157] Carsten Uphoff and Michael Bader. "Yet Another Tensor Toolbox for discontinuous Galerkin methods and other applications". arXiv:1903.11521. Submitted to ACM Transactions on Mathematical Software.

[158] Carsten Uphoff, Sebastian Rettenberger, Michael Bader, Elizabeth H. Madden, Thomas Ulrich, Stephanie Wollherr, and Alice-Agnes Gabriel. "Extreme Scale Multi-physics Simulations of the Tsunamigenic 2004 Sumatra Megathrust Earthquake". In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. SC '17. Denver, Colorado: ACM, 2017, 21:1–21:16. DOI: 10.1145/3126908.3126948.

[159] Peter Vincent, Freddie Witherden, Brian Vermeire, Jin Seok Park, and Arvind Iyer. "Towards Green Aviation with Python at Petascale". In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. SC '16. Salt Lake City, Utah: IEEE Press, 2016, 1:1–1:11. ISBN: 978-1-4673-8815-3. URL: http://dl.acm.org/citation.cfm?id=3014904.3014906.

[160] Peter Wauligmann and Nathan W. Brei. *PSpaMM: Portable Sparse Matrix Multiplication*. https://github.com/peterwauligmann/pspamm. Accessed: 2019-01-21. 2019.

[161] Pauline Weatherall et al. "A new digital bathymetric model of the world's oceans". In: *Earth and Space Science* 2.8 (2015), pp. 331–345.

[162] Lucas C. Wilcox, Georg Stadler, Carsten Burstedde, and Omar Ghattas. "A high-order discontinuous Galerkin method for wave propagation through coupled elastic-acoustic media". In: *Journal of Computational Physics* 229.24 (2010), pp. 9373–9396. DOI: https://doi.org/10.1016/j.jcp.2010.09.008.

[163] Stephanie Wollherr. “Inelastic material response in multi-physics earthquake rupture simulations”. Dissertation. Ludwig-Maximilians-Universität München, forthcoming.

[164] Stephanie Wollherr, Alice-Agnes Gabriel, and Carsten Uphoff. “Off-fault plasticity in three-dimensional dynamic rupture simulations using a modal Discontinuous Galerkin method on unstructured meshes: implementation, verification and application”. In: *Geophysical Journal International* 214.3 (2018), pp. 1556–1584. DOI: 10.1093/gji/ggy213.