A Performance and Energy Study of the Hyperbolic PDE Solver Engine ExaHyPE

Master's Thesis in Computational Science and Engineering

Fabian Güra
Department of Informatics
Technische Universität München

August 31, 2016

Eexaminers: Univ.-Prof. Dr. Michael Bader
Dr. Tobias Weinzierl
Supervisor: Dr. Vasco Varduhn

I hereby declare that this thesis is entirely the result of my own work except where otherwise indicated. I have only used the resources given in the list of references.

Place: ______________ Date: ______________ Signature: ______________________

Abstract

... ...

Contents

Chapter 1

Introduction

- Challenges of exascale
  - Power:
    - cite RAPL Memory: Datacenters are already overprovisioned (worst-case average instead of peak power)
    - cite Energy-aware PAM for HPC: Methods: Direct measurement using specific hardware components (expensive, non-standardized, complex, per package). Software estimates or modeling (inaccurate, finer insight, cite rapl memory: For RAM: sigma = 1.1percent), hybrid
- The ExaHyPE project (numerics, resilience, profiling) as an answer
- On the importance of profiling and performance measuring

Chapter 2

Theory / Theoretical Context / Context

2.1 A D-dimensional ADER-DG scheme with MUSCL-Hancock a-posteriori subcell limiting for non-linear hyperbolic balance laws

2.1.1 Introduction

As stated already in the introduction, the ExaHyPE project is concerned with building an engine for simulating problems that can be formulated in terms of a hyperbolic balance law (HBL). Leaving all of the high performance computing (HPC) components aside, the very heart of the framework therefore comprises of an embedded numerical scheme for solving partial differential equations of HBL type.

Solving PDEs of this kind is of great interest in practice and has been a topic of active research for about a century[1]. A comprehensive overview on schemes that have been proposed over the years can be found in [2]. Two particularly challenging aspects of such simulations are

1. the accurate simulation of HBL problems over long periods of time, especially when facing (prescribed or spontaneously arising) discontinuities ("shocks") or stiff source terms leading to stability issues in space and time, respectively, and,

[1] The British scientist and Durham University alumnus Lewis Fry Richardson is considered to be one of the founding fathers of computational fluid dynamics (CFD). In 1922 he published a book in which he presents a method for weather forecasting based on the solution of differential equations, in a time when most computations were still done by human "computers". See [1] for a second edition of the book published in 2007 which puts the work into a contemporary context and emphasizes how modern weather forecasting is still based on Richardson's ideas.

2. the inherent data access patterns that numerical schemes inherit from the PDE make it challenging to avoid excessive communication and to achieve high arithmetic density, two key drivers for good performance on modern distributed HPC systems.

In ExaHyPE a state of the art method called Arbitrary High Order Derivatives Discontinuous Galerkin (ADER-DG) scheme is employed together with a-posteriori subcell limiting based on the robust, second-order MUSCL-Hancock finite volume method (FVM) scheme (see [3] for details). As the name implies, the Discontinuous Galerkin framework with high order local polynomials ansatz functions acts as the theoretical foundation of the method (see [4, 5] for more details on the underlying theory). In the following sections we will step by step derive the complete scheme and emphasize how it addresses the challenges stated above. The first part on unlimited ADER-DG is based to varying degree on work presented in [6] with three important additional contributions:

1. The scheme is presented in a more general form for systems involving V quantities in D spatial dimensions.
2. We employ index notation and simplify the equations up to a point where the resulting mathematical formulae can easily be mapped to a programming language, in particular it is in direct agreement with FORTRAN code used by Dumbser et al. to generate the numerical results presented in [7].
3. We extend the formulation to include a-posteriori subcell limiting, introduce projection and reconstruction operations to equidistant subgrids and review the MUSCL-Hancock FVM scheme.

The chapter is set up as follows: First, we will begin with some remarks on the notation employed and state the problem at hand in its general form. Second, we will derive an element local weak formulation and approximate it with respect to finite-dimensional function spaces. After introducing reference coordinates, corresponding mappings and orthogonal bases for the function spaces involved, we employ a predictor-corrector approach to arrive at a fully-discrete method that is of arbitrarily high order in both space and time.

2.1.2 Notation

Before we begin deriving the numerical schemes let us quickly introduce the following set of rules on how to depict common mathematical objects and operations:

- Vectors and vector-valued functions, i.e. first-order tensors, will be denoted by bold, lower case letters, e.g. $\boldsymbol{x}$, $\boldsymbol{u}(\boldsymbol{x}, t)$.

- Matrices or matrix-valued functions, i.e. second-order tensors, will be denoted by bold, upper case letters, e.g. $\boldsymbol{K}$, $\boldsymbol{F}(\boldsymbol{x}, t)$.
- Higher order tensors are always denoted as bold, lower-case letters with a "hat" on top, e.g. $\hat{\boldsymbol{u}}^{K,i}$, $\hat{\boldsymbol{q}}^{K,i}$. Note, however, that the opposite is not true[2].
- To avoid confusion in case we deal with tensors for which a superscript or a subscript is "part of its name" or where this indicates membership in a set, similar to the convention in many programming languages we denote "accesses" into the tensor by square brackets. An example illustrating the advantage of this notation based on common naming conventions in literature could be the following: Let $\{\tilde{\boldsymbol{u}}_h^{K,i}\}_{K\in\mathcal{K}_h, i\in\{0,1,\dots,I-1\}}$ be the set of vector-valued functions from an appropriate Hilbert space which are defined locally on a cell $K \in \mathcal{K}_h$ and in a time interval $[t_i, t_i + \Delta t_i]$, $i \in \{0, 1, \dots, I\}$. If we now want to access the v-th component of $\hat{\boldsymbol{u}}^{K,i}$, we write

  $$\left[\tilde{\boldsymbol{u}}_h^{K,i}\right]_v \tag{2.1}$$

  instead of

  $$\tilde{\boldsymbol{u}}_{h,v}^{K,i}. \tag{2.2}$$

  In this way it is absolutely clear that K, i and h are "part of the name" and that v is an index used to access an element in the tensor. However in case an expression is absolutely unambiguous, for the sake of brevity we will often omit the square brackets. Most prominently we write

  $$\frac{\partial}{\partial x_d} \tag{2.3}$$

  instead of

  $$\frac{\partial}{\partial \left[\boldsymbol{x}\right]_d}. \tag{2.4}$$

- Throughout the thesis we use index notation following the Einstein summation convention whenever possible. This means that if an index within a product expression is repeated exactly once, this implies summation over the whole range of this index. The standard inner product of two vectors $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^N$ can then be written as

  $$\langle \boldsymbol{x}, \boldsymbol{y} \rangle = \sum_{n=0}^{N-1} [\boldsymbol{x}]_n \, [\boldsymbol{y}]_n = [\boldsymbol{x}]_n \, [\boldsymbol{y}]_n \, (= [\boldsymbol{x}]_m \, [\boldsymbol{y}]_m) \tag{2.5}$$

[2] In general the dimensionality of an objects will always be obvious from its indices. Since we will just allow scalar and vector-valued indices only this distinction is of critical importance.

and for unambiguous cases like above

$$\langle \boldsymbol{x}, \boldsymbol{y} \rangle = \sum_{n=0}^{N-1} x_n y_n := x_n y_n (= x_m y_m). \tag{2.6}$$

Such indices are called dummy indices in the sense that as illustrated in the example above it does not matter if the index is named n or m. Sometimes free indices, i.e. indices that are not dummy indices, appear twice in a term as a result of some algebraic manipulation. In these cases we will explicitly state that summation over the index is not intended, unless it is obvious e.g. from the left-hand side of the equation that the index can only be a free index. We furthermore always give an explicit range for free indices. See [8] for more details on index notation, its advantages and disadvantages as well as a more formal definition.

In addition to increased brevity, index notation allows for less ambiguities compared to classical vector notation, simplifies derivation of identities from tensor calculus and if done carefully the resulting formulae can be conveniently mapped to loops in low-level programming languages such as C or FORTRAN.

- To keep all derivations dimension-agnostic, we define accesses into tensors using vector indices as follows: Let $\hat{\boldsymbol{u}} \in \mathbb{R}^{I_1 \times I_2 \times \dots \times I_D}$ be a tensor of order $D \in \mathbb{N}$ with $I_d \in \mathbb{N}_0$ for all $d \in \{0, 1, \dots, D-1\} := \mathcal{D}$. Let furthermore $i_d \in \{0, 1, \dots, I_d - 1\}$ for all $d \in \mathcal{D}$ and $\boldsymbol{i} \in \mathbb{N}_0^D$, $[\boldsymbol{i}]_d = i_d$ for $d \in \mathcal{D}$ a vector of indices. Then we define

$$[\hat{\boldsymbol{u}}]_{\boldsymbol{i}} = [\hat{\boldsymbol{u}}]_{[i_0, i_1, \dots, i_{D-1}]} = [\hat{\boldsymbol{u}}]_{i_0, i_1, \dots, i_{D-1}}. \tag{2.7}$$

  If we only provide a vector of $D-1$ indices, i.e.

$$[\hat{\boldsymbol{u}}]_{[i_0, i_1, \dots, i_{D-2}]}, \tag{2.8}$$

  we obtain a vector of length I_{D-1}. If we only provide $D-2$ indices we obtain a matrix with I_{D-2} rows and I_{D-1} columns. In general if we provide $d \in \{0, 1, \dots, D\}$ indices we obtain a tensor of order $D - d$.

- In the style of numerical computing environments such as MATLAB® or Octave (see [9]) we define the following shorthand notation for sequences of consecutive integral numbers:

$$j{:}k := \begin{cases} \{j, j+1, \dots, k\} & \text{if } j \leq k \\ \{\} & \text{otherwise.} \end{cases} \tag{2.9}$$

- We can now define access into a vector $\boldsymbol{x} \in \mathbb{R}^N$ of length N via sequences as

$$[\boldsymbol{x}]_{j:k} := \left[[\boldsymbol{x}]_j, [\boldsymbol{x}]_{j+1}, \dots, [\boldsymbol{x}]_k \right] \tag{2.10}$$

for $j \leq k$ and $j, k \in 0\!:\!N-1$, which for unambiguous cases as above is equal to the definition

$$\boldsymbol{x}_{j:k} := [x_j, j_{j+1}, \ldots, x_k]. \tag{2.11}$$

Together with implicit set and vector concatenation we can then write for $k \in 0 : N-1$

$$[\boldsymbol{x}]_{\{0:k-1,k+1:N-1\}} = \boldsymbol{x}_{\{0:k-1,k+1:N-1\}} \tag{2.12}$$

to denote the vector of length $N-1$ that contains all values of the original vector $\boldsymbol{x}$ but the k-th component. Furthermore

$$\Big[[\boldsymbol{x}]_{0:k-1}, x', [\boldsymbol{x}]_{k+1:N-1}\Big] = \Big[\boldsymbol{x}_{0:k-1}, x', \boldsymbol{x}_{k+1:N-1}\Big] \tag{2.13}$$

denotes the vector of length N whose components are equal to the ones of $\boldsymbol{x}$ apart from the k-th one, which we have replaced by the scalar $x' \in \mathbb{R}$.

2.1.3 Hyperbolic Balance Laws

A D-dimensional balance law in a system with V quantities is described mathematically by a partial differential equation (PDE) of the form

$$\frac{\partial}{\partial t}\Big[\boldsymbol{u}(\boldsymbol{x},t)\Big]_v + \frac{\partial}{\partial x_d}\Big[\boldsymbol{F}\left(\boldsymbol{u}(\boldsymbol{x},t)\right)\Big]_{vd} = \Big[\boldsymbol{s}\left(\boldsymbol{u}(\boldsymbol{x},t)\right)\Big]_v \text{ on } \boldsymbol{\Omega} \times [0,T] \tag{2.14}$$

together with initial conditions

$$\big[\boldsymbol{u}(\boldsymbol{x},0)\big]_v = \big[\boldsymbol{u}_0(\boldsymbol{x})\big]_v \quad \forall \boldsymbol{x} \in \boldsymbol{\Omega}, \tag{2.15}$$

and boundary conditions

$$\big[\boldsymbol{u}(\boldsymbol{x},t)\big]_v = \big[\boldsymbol{u}_B(\boldsymbol{x},t)\big]_v \quad \forall \boldsymbol{x} \in \partial\boldsymbol{\Omega}, t \in [0,T], \tag{2.16}$$

for all $v \in \mathcal{V}$, where we define the index set $\mathcal{V} = \{1, 2, \ldots, V\}$. $[0,T]$ is the time interval of interest and $\boldsymbol{\Omega} \subset \mathbb{R}^D$ denotes the spatial domain. The function $\boldsymbol{F} : \mathbb{R}^V \to \mathbb{R}^{V\times D}, \boldsymbol{u} \mapsto \boldsymbol{F}(\boldsymbol{u}) = \big[\boldsymbol{f}_1(\boldsymbol{u}), \boldsymbol{f}_2(\boldsymbol{u}), \ldots, \boldsymbol{f}_D(\boldsymbol{u})\big]$ is called the flux function. For the problem to be hyperbolic we require that all Jacobian matrices $\boldsymbol{A}_d(\boldsymbol{u})$, $d \in \{0, 1, \ldots, D-1\} := \mathcal{D}$, defined as

$$[\boldsymbol{A}_d]_{ij} = \frac{\partial\, [\boldsymbol{f}_d]_i}{\partial u_j}, \tag{2.17}$$

have D real eigenvalues in each admissible state $\boldsymbol{u} \in \mathbb{R}^V$.

2.1.4 Space and Time Discretization

Let $\mathcal{K}_h$ be a quadrilateral partition of $\mathbf{\Omega}$, i.e.

$$K \cap J = \emptyset \forall K, J \in \mathcal{K}_h, K \neq J, \tag{2.18}$$

$$\bigcup_{K \in \mathcal{K}_h} K = \mathbf{\Omega}. \tag{2.19}$$

For the index set $\mathcal{I} := \{0, 1, \ldots, I-1\}$ let $\{t_i\}_{i \in \mathcal{I}}$ be an I-fold partition of the time interval $[0, T]$ such that

$$0 = t_0 < t_1 < \ldots < t_I = T. \tag{2.20}$$

For $i \in \mathcal{I}$ we furthermore define

$$\Delta t_i = t_{i+1} - t_i, \tag{2.21}$$

so that the subinterval $[t_i, t_{i+1}]$ can be written as $[t_i, t_i + \Delta t_i]$.

Without loss of generality we can solve the original PDE (2.14) on $\mathbf{\Omega} \times [0, T]$ simply by solving the PDE locally for each element $K \in \mathcal{K}_h$ in the time interval $[t_0, t_0 + \Delta t_0]$ and then proceeding to the next time interval until we have reached the final time T. This gives rise to an element-local formulation on a subinterval in time which we will focus in the following.

2.1.5 Element-local Weak Formulation

Let $L^2(\mathbf{\Omega})^V$ be the space of vector-valued, square-integrable functions on $\mathbf{\Omega}$, i.e.

$$L^2(\mathbf{\Omega})^V = \left\{ \boldsymbol{w} : \mathbf{\Omega} \to \mathbb{R}^V \mid \int_{\mathbf{\Omega}} \|\boldsymbol{w}\|^2 \, d\boldsymbol{x} < \infty \right\}. \tag{2.22}$$

Let $\boldsymbol{w} \in L^2(\mathbf{\Omega})^V$ be a spatial test function. Multiplication of the original PDE (2.14) and integration over a space-time cell $K \times [t_i, t_i + \Delta t_i]$ yields a element-local weak formulation of the problem,

$$\int_{t_i}^{t_i+\Delta t_i} \int_K \frac{\partial}{\partial t} [\boldsymbol{u}]_v [\boldsymbol{w}]_v \, d\boldsymbol{x} dt + \int_{t_i}^{t_i+\Delta t_i} \int_K \frac{\partial}{\partial x_d} \left[\boldsymbol{F}(\boldsymbol{u})\right]_{vd} [\boldsymbol{w}]_v \, d\boldsymbol{x} dt =$$
$$\int_{t_i}^{t_i+\Delta t_i} \int_K \left[\boldsymbol{s}(\boldsymbol{u})\right]_v [\boldsymbol{w}]_v \, d\boldsymbol{x} dt, \tag{2.23}$$

which we require to hold for all $v \in \mathcal{V}$, $\boldsymbol{w} \in L^2(\mathbf{\Omega})^V$, $K \in \mathcal{K}_h$ and $i \in \mathcal{I}$.

Integration by parts of the spatial integral in the second term yields

$$\int_K \frac{\partial}{\partial x_d} \left[\boldsymbol{F}(\boldsymbol{u})\right]_{vd} [\boldsymbol{w}]_v \, d\boldsymbol{x} =$$
$$\int_K \frac{\partial}{\partial x_d} \left(\left[\boldsymbol{F}(\boldsymbol{u})\right]_{vd} [\boldsymbol{w}]_v \right) d\boldsymbol{x} - \int_K \left[\boldsymbol{F}(\boldsymbol{u})\right]_{vd} \frac{\partial}{\partial x_d} [\boldsymbol{w}]_v \, d\boldsymbol{x}. \tag{2.24}$$

Application of the divergence theorem to the first term on the right-hand side of (2.24) yields

$$\int_K \frac{\partial}{\partial x_d} \left(\left[\boldsymbol{F}(\boldsymbol{u}) \right]_{vd} \left[\boldsymbol{w} \right]_v \right) \, dx = \int_{\partial K} \left[\boldsymbol{F}(\boldsymbol{u}) \right]_{vd} \left[\boldsymbol{w} \right]_v \left[\boldsymbol{n} \right]_d \, ds(\boldsymbol{x}), \tag{2.25}$$

where $\boldsymbol{n} \in \mathbb{R}^D$ is the unit-length, outward-pointing normal vector at a point $\boldsymbol{x}$ on the surface of K, which we denote by ∂K.

Inserting eqs. (2.24) and (2.25) into eq. (2.23) yields the following more favorable element-local weak formulation of the original equation (2.14):

$$\begin{aligned} &\int_{t_i}^{t_i+\Delta t_i} \int_K \frac{\partial}{\partial t} \left[\boldsymbol{u} \right]_v \left[\boldsymbol{w} \right]_v \, dx dt - \int_{t_i}^{t_i+\Delta t_i} \int_K \left[\boldsymbol{F}(\boldsymbol{u}) \right]_{vd} \frac{\partial}{\partial x_d} \left[\boldsymbol{w} \right]_v \, dx dt + \\ &\int_{t_i}^{t_i+\Delta t_i} \int_{\partial K} \left[\boldsymbol{F}(\boldsymbol{u}) \right]_{vd} \left[\boldsymbol{w} \right]_v \left[\boldsymbol{n} \right]_d \, ds(\boldsymbol{x}) dt = \int_{t_i}^{t_i+\Delta t_i} \int_K \left[\boldsymbol{s}(\boldsymbol{u}) \right]_v \left[\boldsymbol{w} \right]_v \, dx dt. \end{aligned} \tag{2.26}$$

Again we require the weak formulation to hold for all $v \in \mathcal{V}$, $\boldsymbol{w} \in L^2(\boldsymbol{\Omega})^V$, $K \in \mathcal{K}_h$ and $i \in \mathcal{I}$.

2.1.6 Restriction to Finite-Dimensional Function Spaces

To discretize eq. (2.26) we need to impose the restriction that both test and ansatz functions come from a finite-dimensional function space. First, let $\mathbb{Q}_N(K)^V$ and $\mathbb{Q}_N(K \times [t_i, t_i + \Delta t_i])^V$ be the space of vector-valued, multivariate polynomials of degree less or equal than N in each variable on K and $K \times [t_i, t_i + \Delta t_i]$, respectively. We can then define the following finite-dimensional function spaces:

- For spatial functions we define

$$\mathbb{W}_h = \left\{ \boldsymbol{w}_h \in L^2(\boldsymbol{\Omega})^V \mid \boldsymbol{w}_h|_K := \boldsymbol{w}_h^K \in \mathbb{Q}_N(K)^V \; \forall K \in \mathcal{K}_h \right\}. \tag{2.27}$$

- For space-time functions on the time subinterval $[t_i, t_i + \Delta t_i]$, $i \in \mathcal{I}$ we define

$$\begin{aligned} \tilde{\mathbb{W}}_h^i = \Big\{ & \tilde{\boldsymbol{w}}_h^i \in L^2 \left(\boldsymbol{\Omega} \times [t_i, t_i + \Delta t_i] \right) \mid \\ & \tilde{\boldsymbol{w}}_h^i \Big|_K := \tilde{\boldsymbol{w}}_h^{K,i} \in \mathbb{Q}_N \left(K \times [t_i, t_i + \Delta t_i] \right) \; \forall K \in \mathcal{K}_h \Big\}. \end{aligned} \tag{2.28}$$

Replacing $\boldsymbol{w}$ by $\boldsymbol{w}_h \in \mathbb{W}_h$ and $\boldsymbol{u}$ by $\tilde{\boldsymbol{u}}_h^i \in \tilde{\mathbb{W}}_h^i$ in eq. (2.26), i.e. restricting ourselves to test and ansatz functions from finite-dimensional function spaces, yields an approximation of the weak formulation,

$$\int_{t_i}^{t_i+\Delta t_i} \int_K \frac{\partial}{\partial t} \left[\tilde{\boldsymbol{u}}_h^{K,i}\right]_v \left[\boldsymbol{w}_h^K\right]_v d\boldsymbol{x}dt - \int_{t_i}^{t_i+\Delta t_i} \int_{\partial K} \left[\boldsymbol{F}(\tilde{\boldsymbol{u}}_h^{K,i})\right]_{vd} \frac{\partial}{\partial x_d} \left[\boldsymbol{w}_h^K\right]_v d\boldsymbol{x}dt +$$
$$\int_{t_i}^{t_i+\Delta t_i} \int_{\partial K} \left[\boldsymbol{\mathcal{G}}(\tilde{\boldsymbol{u}}_h^{K,i}, \tilde{\boldsymbol{u}}_h^{K^+ i}, \boldsymbol{n})\right]_v \left[\boldsymbol{w}_h^K\right]_v ds(\boldsymbol{x})dt =$$
$$\int_{t_i}^{t_i+\Delta t_i} \int_K \left[\boldsymbol{s}(\tilde{\boldsymbol{u}}_h^{K,i})\right]_v \left[\boldsymbol{w}_h^K\right]_v d\boldsymbol{x}dt, \tag{2.29}$$

which now has to hold for all $\boldsymbol{w}_h \in \mathbb{W}_h$, $K \in \mathcal{K}_h$ and $i \in \mathcal{I}$. Since for a cell $K \in \mathcal{K}_h$ and one of its Voronoi neighbors $K' \in V(K)$ in general it holds that

$$\tilde{\boldsymbol{u}}_h^{K,i}(\boldsymbol{x}^*) \neq \tilde{\boldsymbol{u}}_h^{K'i}(\boldsymbol{x}^*) \tag{2.30}$$

for $\boldsymbol{x}^* \in K \cap K'$, i.e. $\tilde{\boldsymbol{u}}_h^i$ is double-valued at the interface between K and K', in order to compute the surface integral we need to introduce the numerical flux function $\boldsymbol{\mathcal{G}}(\tilde{\boldsymbol{u}}_h^{K,i}, \tilde{\boldsymbol{u}}_h^{K'i}, \boldsymbol{n})$. The numerical flux at a position $\boldsymbol{x}^* \in K \cap K'$ on the interface is obtained by (approximately) solving a Riemann problem in normal direction.

Excursus: The Riemann Problem

Let $\boldsymbol{x}^*$ be a point on interface ∂K between a cell $K \in \mathcal{K}_h$ and its Voronoi neighbor $K' \in V(K)$ and let $\boldsymbol{n}$ be the outward pointing unit normal vector at this point. Then to obtain the numerical flux we need to solve the initial boundary value problem ("Riemann problem")

$$\frac{\partial}{\partial t} [\boldsymbol{g}]_v + \sum_{d=1}^{D} \frac{\partial}{\partial x_d} [\boldsymbol{F}(\boldsymbol{g})]_{vd} [\boldsymbol{n}]_d = 0 \tag{2.31}$$

along the line $\boldsymbol{x} = \boldsymbol{x}^* + \alpha \boldsymbol{n}$ for $\alpha \in \mathbb{R}$ with discontinuous initial conditions

$$\boldsymbol{g}(\boldsymbol{x}^* + \alpha\boldsymbol{n}, 0) = \begin{cases} \tilde{\boldsymbol{u}}_h^{K,i}\Big|_{\boldsymbol{x}^*} & \text{if } \alpha < 0 \\ \tilde{\boldsymbol{u}}_h^{K',i}\Big|_{\boldsymbol{x}^*} & \text{if } \alpha > 0. \end{cases} \tag{2.32}$$

We then evaluate the similarity solution $\tilde{\boldsymbol{g}}(\alpha/t)$ of the problem and define

$$\left[\boldsymbol{\mathcal{G}}\left(\tilde{\boldsymbol{u}}_h^{K,i}, \tilde{\boldsymbol{u}}_h^{K',i}, \boldsymbol{n}\right)\right]_v := \left[\tilde{\boldsymbol{g}}\big|_0\right]_v. \tag{2.33}$$

For an extensive overview on state of the art approximate Riemann solvers see [2].

Continuing with eq. (2.29), integration by parts in time of the first term and noting that $\boldsymbol{w}_h$ is constant in time yields the following one-step update scheme for the cell-local time-discrete solution $\tilde{\boldsymbol{u}}_h^{K,i}$:

$$\begin{aligned}\int_K \left[\tilde{\boldsymbol{u}}_h^{K,i}\Big|_{t_i+\Delta t_i}\right]_v \left[\boldsymbol{w}_h^K\right]_v d\boldsymbol{x} = \int_K \left[\tilde{\boldsymbol{u}}_h^{K,i}\Big|_{t_i}\right]_v \left[\boldsymbol{w}_h^K\right]_v d\boldsymbol{x} + &\\ \int_{t_i}^{t_i+\Delta t_i}\int_K \left[\boldsymbol{F}(\tilde{\boldsymbol{u}}_h^{K,i})\right]_{vd} \frac{\partial}{\partial x_d}\left[\boldsymbol{w}_h^K\right]_v d\boldsymbol{x}dt - &\\ \int_{t_i}^{t_i+\Delta t_i}\int_{\partial K} \left[\boldsymbol{\mathcal{G}}(\tilde{\boldsymbol{u}}_h^{K,i}, \tilde{\boldsymbol{u}}_h^{K^+i}, \boldsymbol{n})\right]_v \left[\boldsymbol{w}_h^K\right]_v ds(\boldsymbol{x})dt + &\\ \int_{t_i}^{t_i+\Delta t_i}\int_K \left[\boldsymbol{s}(\tilde{\boldsymbol{u}}_h^{K,i})\right]_v \left[\boldsymbol{w}_h^K\right]_v d\boldsymbol{x}dt. &\end{aligned} \tag{2.34}$$

Again we require eq. (2.34) to hold for all $v \in \mathcal{V}$, $\boldsymbol{w}_h \in \mathbb{W}_h$, $K \in \mathcal{K}_h$ and $i \in \mathcal{I}$. Note, however, that the scheme is incomplete, since we only know $\tilde{\boldsymbol{u}}_h^i|_t$ at the discrete time steps $t \in \{t_i, t_i + \Delta t_i\}$, not within the open interval, i.e. for $t \in (t_i, t_i + \Delta t_i)$. As commonly done in a DG framework we therefore proceed by replacing $\tilde{\boldsymbol{u}}_h$ on the interval $(t_i, t_i + \Delta t_i)$ by an approximation $\tilde{\boldsymbol{q}}_h^i \in \tilde{\mathbb{W}}_h^i$ which we call space-time predictor.

2.1.7 Space-time Predictor

To derive a procedure to compute the space-time predictor $\tilde{\boldsymbol{q}}_h^i \in \tilde{\mathbb{W}}_h^i$ we again start from the original PDE (2.14), but this time we do not use a spatial test function $\boldsymbol{w}_h \in \mathbb{W}_h$, but a space-time test function $\tilde{\boldsymbol{w}}_h^i \in \tilde{\mathbb{W}}_h^i$. If we furthermore replace the solution $\boldsymbol{u}$ by the space-time predictor $\tilde{\boldsymbol{q}}_h^i \in \tilde{\mathbb{W}}_h^i$, integrate over the space-time element $K \times [t_i, t_i + \Delta t_i]$ and apply the divergence theorem analogously to eq. (2.25) we obtain the following relation:

$$\begin{aligned}&\int_{t_i}^{t_i+\Delta t_i}\int_K \frac{\partial}{\partial t}\left[\tilde{\boldsymbol{q}}_h^{K,i}\right]_v \left[\tilde{\boldsymbol{w}}_h^{K,i}\right]_v d\boldsymbol{x}dt - \\ &\int_{t_i}^{t_i+\Delta t_i}\int_K \left[\boldsymbol{F}(\tilde{\boldsymbol{q}}_h^{K,i})\right]_{vd} \frac{\partial}{\partial x_d}\left[\tilde{\boldsymbol{w}}_h^{K,i}\right]_v d\boldsymbol{x}dt + \\ &\int_{t_i}^{t_i+\Delta t_i}\int_{\partial K} \left[\boldsymbol{\mathcal{G}}\left(\tilde{\boldsymbol{q}}_h^{K,i}, \tilde{\boldsymbol{q}}_h^{K^+i}, \boldsymbol{n}\right)\right]_v \left[\tilde{\boldsymbol{w}}_h^{K,i}\right]_v ds(\boldsymbol{x})dt = \\ &\int_{t_i}^{t_i+\Delta t_i}\int_K \left[\boldsymbol{s}\left(\tilde{\boldsymbol{q}}_h^{K,i}\right)\right]_v \left[\tilde{\boldsymbol{w}}_h^{K,i}\right]_v d\boldsymbol{x}dt.\end{aligned} \tag{2.35}$$

We require eq. (2.35) to hold for all $v \in \mathcal{V}$, $\tilde{\boldsymbol{w}}_h^i \in \tilde{\mathbb{W}}_h^i$, $K \in \mathcal{K}_h$ and $i \in \mathcal{I}$.

The assumption that the solution is balanced, i.e. that there is no net inflow or outflow for cells $K \in \mathcal{K}_h$ allows us to drop the third term. Together with integration by parts in time applied to the first term this yields

$$\begin{aligned}&\int_K \left[\tilde{\boldsymbol{q}}_h^{K,i}\Big|_{t_i+\Delta t_i} \right]_v \left[\tilde{\boldsymbol{w}}_h^{K,i}\Big|_{t_i+\Delta t_i} \right]_v d\boldsymbol{x} - \int_{t_i}^{t_i+\Delta t_i}\int_K \left[\tilde{\boldsymbol{q}}_h^{K,i}\right]_v \frac{\partial}{\partial t} \left[\tilde{\boldsymbol{w}}_h^{K,i}\right]_v d\boldsymbol{x}dt = \\ &\int_K \left[\tilde{\boldsymbol{q}}_h^{K,i}\Big|_{t_i} \right]_v \left[\tilde{\boldsymbol{w}}_h^{K,i}\Big|_{t_i} \right]_v d\boldsymbol{x} + \int_{t_i}^{t_i+\Delta t_i}\int_K \left[\boldsymbol{F}(\tilde{\boldsymbol{q}}_h^{K,i})\right]_{vd} \frac{\partial}{\partial x_d} \left[\tilde{\boldsymbol{w}}_h^{K,i}\right]_v d\boldsymbol{x}dt + \\ &\int_{t_i}^{t_i+\Delta t_i}\int_K \left[\boldsymbol{s}\left(\tilde{\boldsymbol{q}}_h^{K,i}\right)\right]_v \left[\tilde{\boldsymbol{w}}_h^{K,i}\right]_v d\boldsymbol{x}dt,\end{aligned} \tag{2.36}$$

which we require to hold for all $v \in \mathcal{V}$, $\tilde{\boldsymbol{w}}_h^i \in \tilde{\mathbb{W}}_h^i$, $K \in \mathcal{K}_h$ and $i \in \mathcal{I}$. In conjunction with the initial condition

$$\tilde{\boldsymbol{q}}_h^{K,i}\Big|_{t_i} = \tilde{\boldsymbol{u}}_h^{K,i} \tag{2.37}$$

and an initial guess

$$\tilde{\boldsymbol{q}}_h^{K,i}\Big|_{t} = \tilde{\boldsymbol{u}}_h^{K,i} \,\forall t \in (t_i, t_i + \Delta t_i] \tag{2.38}$$

this relation can be used as a fixed-point iteration to find the cell-local space-time predictor $\tilde{\boldsymbol{q}}_h^{K,i}$.

In the following two sections we will introduce mappings from spatial elements K and space-time elements $K \times [t_i, t_i + \Delta t_i]$ to spatial and space-time reference cells and orthogonal bases for the spaces $\mathbb{W}_h$ and $\tilde{\mathbb{W}}_h^i$. We will then insert these results into eq. (2.36) and derive a fully-discrete iterative method to compute the cell-local space-time predictor $\tilde{\boldsymbol{q}}_h^{K,i}$.

2.1.8 Reference Elements and Mappings

Let $\hat{K} := [0,1]^D$ be the spatial reference element and $\boldsymbol{\xi} \in \hat{K}$ be a point therein. Let $[0,1]$ be the reference time interval and $\tau \in [0,1]$ be a point reference time. We can then introduce the following mappings:

Spatial mappings: Let $K \in \mathcal{K}_h$ be a cell in global coordinates with extent $\Delta\boldsymbol{x}^K$ and "lower-left corner" $\boldsymbol{P}_K$, more precisely that is

$$\left[\Delta\boldsymbol{x}^K\right]_d = \max_{\boldsymbol{x}\in K}\,[\boldsymbol{x}]_d - \min_{\boldsymbol{x}\in K}\,[\boldsymbol{x}]_d \tag{2.39}$$

and

$$[\boldsymbol{P}_K]_d = \min_{\boldsymbol{x}\in K}\,[\boldsymbol{x}]_d \tag{2.40}$$

for $d \in \mathcal{D}$. We can then define a mapping

$$\boldsymbol{\mathcal{X}}_K : \hat{K} \to K, \boldsymbol{\xi} \mapsto \boldsymbol{\mathcal{X}}_K(\boldsymbol{\xi}) = \boldsymbol{x} \tag{2.41}$$

via the relation

$$[\boldsymbol{x}]_d = \left[\boldsymbol{\mathcal{X}}_K(\boldsymbol{\xi})\right]_d = [\boldsymbol{P}_K]_d + [\Delta\boldsymbol{x}]_d \left[\boldsymbol{\xi}\right]_d \tag{2.42}$$

for $d \in \mathcal{D}$ (i.e. no summation on d) and for all $\boldsymbol{x} \in K$, $\boldsymbol{\xi} \in \hat{K}$ and $K \in \mathcal{K}_h$.

Temporal mappings: Let $[t_i, t_i + \Delta t_i], i \in \mathcal{I}$ be an interval in global time. The mapping

$$\mathcal{T}_i : [0,1] \to [t_i, t_i + \Delta t_i], \tau \mapsto \mathcal{T}_i(\tau) = t_i + \Delta t_i \tau = t \tag{2.43}$$

maps a point $\tau \in [0,1]$ in reference time to a point $t \in [t_i, t_i + \Delta t_i]$ in global time for all $i \in \mathcal{I}$.

The inverse mappings, the Jacobian matrices and the Jacobi determinants of the mappings are given in the following:

Spatial mappings: The inverse spatial mappings

$$\boldsymbol{\mathcal{X}}_K^{-1} : K \to \hat{K}, \boldsymbol{x} \mapsto \boldsymbol{\mathcal{X}}_K^{-1}(\boldsymbol{x}) = \boldsymbol{\xi} \tag{2.44}$$

are defined via the relation

$$[\boldsymbol{\xi}]_d = \left[\boldsymbol{\mathcal{X}}_K^{-1}(\boldsymbol{x})\right]_d = \frac{1}{\left[\Delta\boldsymbol{x}^K\right]_d}\left([\boldsymbol{x}]_d - [\boldsymbol{P}_K]_d\right) \tag{2.45}$$

for $d \in \mathcal{D}$ and for all $\boldsymbol{\xi} \in \hat{K}$, $\boldsymbol{x} \in K$ and $K \in \mathcal{K}_h$. The Jacobian of $\boldsymbol{\mathcal{X}}_K$ is found to be

$$\left[\frac{\partial \boldsymbol{\mathcal{X}}_K}{\partial \boldsymbol{\xi}}\right]_{dd'} = \frac{\partial \left[\boldsymbol{\mathcal{X}}_K\right]_d}{\partial \xi_{d'}} = \left[\Delta\boldsymbol{x}^K\right]_d \delta_{dd'}, \tag{2.46}$$

where $d, d' \in \mathcal{D}$ (i.e. no summation on d) and for all $K \in \mathcal{K}_h$. As usual $\delta_{dd'}$ denotes the Kronecker delta defined as

$$\delta_{dd'} = \begin{cases} 0 & \text{if } d \neq d' \\ 1 & \text{if } d = d'. \end{cases} \tag{2.47}$$

The Jacobi determinant of $\boldsymbol{\mathcal{X}}_K$ for $K \in \mathcal{K}_h$ then simply is

$$J_{\boldsymbol{\mathcal{X}}_K} = \|\frac{\partial \boldsymbol{\mathcal{X}}_K}{\partial \boldsymbol{\xi}}\| = \prod_{d=1}^{D} \left[\Delta\boldsymbol{x}^K\right]_d, \tag{2.48}$$

i.e. the determinant is constant for all $\boldsymbol{\xi} \in \hat{K}$.

Temporal mappings: The inverse temporal mappings are given as

$$\mathcal{T}_i^{-1} : [t_i, t_i + \Delta t_i] \to [0,1], t \mapsto \mathcal{T}_i^{-1}(t) = \frac{t - t_i}{\Delta t_i} = \tau \tag{2.49}$$

for all $\tau \in [0,1]$, $t \in [t_i, t_i + \Delta t_i]$ and $i \in \mathcal{I}$. In the trivial case of a one-dimensional mapping the Jacobian of $\mathcal{T}_i$ is a scalar which in turn is its own determinant. One finds

$$\frac{d\mathcal{T}_i}{\partial \tau} = \Delta t_i = J_{\mathcal{T}_i} \tag{2.50}$$

which again is constant for all $\tau \in [0,1]$.

2.1.9 Orthogonal Bases for the Finite-dimensional Function Spaces

In section 2.1.6 we introduced finite-dimensional, cell-wise polynomial function spaces $\mathbb{W}_h$ and $\tilde{\mathbb{W}}_h^i$ for spatial and space-time ansatz and test functions, respectively. On our way towards a fully discrete version of the relations (2.36) and (2.34) to obtain the space-time predictor and the solution at the next time step, respectively, we will now derive a set of functions that form bases for the two function spaces of interest. Following the approach presented by Dumbers et al. in **??**, throughout the thesis we will use the set of Lagrange functions with nodes located at the roots of the Legendre polynomials and tensor products thereof. In the later chapters of this work it will become obvious why this particular choice is highly favorable. For the moment the two major reasons shall be stated as an outlook:

1. Numerical integration using the Gauss-Legendre method is simple and computationally cheap, since the function values at the Gauss-Legendre nodes are directly available as they are equal to the degrees of freedom representing the local polynomial.
2. The resulting bases are orthogonal, which in turn makes sure that the resulting DG-matrices exhibit a spares block structure allowing computations to be carried out efficiently in a dimension-by-dimension manner.

Lagrange Interpolation

Let $f \in \mathbb{Q}_N([0,1])$ be a polynomial of degree less or equal than N and for the index set $\mathcal{N} := \{0, 1, \ldots, N\}$ let $\{\hat{\xi}_n\}_{n \in \mathcal{N}}$ be a set of distinct nodes in $[0,1]$. Then the Lagrange interpolation of f,

$$\hat{f}(\xi) = \sum_{n=0}^{N} L_n(\xi) f(\xi_n) \tag{2.51}$$

with Lagrange functions

$$L_n(\xi) = \prod_{m=0,m\neq n}^{N} \frac{\xi - \hat{\xi}_m}{\hat{\xi}_n - \hat{\xi}_m} \tag{2.52}$$

is exact, i.e.

$$f(\xi) = \hat{f}(\xi) \quad \forall \xi \in [0,1]. \tag{2.53}$$

Since therefore every polynomial $f \in \mathbb{Q}_N([0,1])$ can be represented as a linear combination of the Legendre polynomials L_n, $n \in \mathcal{N}$, the set of functions $\{L_n\}_{n\in\mathcal{N}}$ is a basis of $Q_N([0,1])$.

The following observation is an important property of the Lagrange polynomials:

$$L_n(\hat{\xi}_{n'}) = \delta_{nn'}, \tag{2.54}$$

i.e. at each node $\hat{\xi}_n$ only L_n has value 1 and all other polynomials evaluate to 0.

Legendre Polynomials and Gauss-Legendre Integration

Let $P_0 : [-1,1] \to \mathbb{R}, \xi \mapsto 1$ and $P_1 : [-1,1] \to \mathbb{R}, \xi \mapsto \xi$ be the zeroth and the first Legendre polynomial, respectively. Then the $N+1$-st Legendre polynomial can be defined via the following recurrence relation:

$$P_{N+1}(\xi) = \frac{1}{N+1}\left((2N+1)P_N(\xi) - nP_{N-1}(\xi)\right). \tag{2.55}$$

Let $\{\tilde{\xi}_n\}_{n\in\mathcal{N}}$ be the roots of the $N+1$-st Legendre polynomial L_{N+1}. Then $\{\hat{\xi}_n\}_{n\in\mathcal{N}}$ with

$$\hat{\xi}_n = \frac{1}{2}(\tilde{\xi}_n + 1) \tag{2.56}$$

are the roots of the $N+1$-st Legendre polynomial linearly mapped to the interval $(0,1)$. In conjunction with a set of suitable weights $\{\hat{\omega}_n\}_{n\in\mathcal{N}}$ Gauss-Legendre integration can be used to integrate polynomials of degree up to $2N+1$ over the integral $[0,1]$ exactly, i.e.

$$\int_0^1 f(\xi)\, d\xi = \sum_{n=0}^{N} \hat{\omega}_n f(\hat{\xi}_n) \quad \forall f \in \mathbb{Q}_{2N+1}\left([0,1]\right). \tag{2.57}$$

A Python notebook on how to find the nodes $\{\hat{\xi}_n\}_{n\in\mathcal{N}}$ and weights $\{\hat{\omega}_n\}_{n\in\mathcal{N}}$ can be found in appendix A.

Scalar-valued Basis Functions on the One-dimensional Reference Element

Let $\{\hat{\psi}_n\}_{n \in \mathcal{N}}$ be the set of $N+1$ Lagrange polynomials with nodes at the roots of the $N+1$-st Legendre polynomial linearly mapped to the interval $[0,1]$, i.e.

$$\hat{\psi}_n(x) = \sum_{n'=0}^{N} \frac{x - \hat{x}_{n'}}{\hat{x}_n - \hat{x}_{n'}} \tag{2.58}$$

for $n \in \mathcal{N}$. Since $\{\hat{\psi}_n\}_{n \in \mathcal{N}}$ are Lagrange polynomials and the roots $\{\hat{x}_n\}_{n \in \mathcal{N}}$ are distinct the set is a basis of $\mathbb{Q}_N([0,1])$. Since furthermore

$$\left\langle \hat{\psi}_n, \hat{\psi}_m \right\rangle_{L^2\left([0,1]\right)} = \int_0^1 \hat{\psi}_n(x)\hat{\psi}_m(x)\, d\boldsymbol{x} = \sum_{n'=0}^{N} \hat{\varpi}'_n \hat{\psi}_n(\hat{x}_{n'})\hat{\psi}_m(\hat{x}_{n'}) = \hat{\varpi}_n \delta_{mn} \tag{2.59}$$

for all $m, n \in \mathcal{N}$ (i.e. no summation over n), the set is even an orthogonal basis of $\mathbb{Q}_N([0,1])$ with respect to the L^2-scalar product as defined above. In this derivation we used the fact that $\hat{\psi}_n\hat{\psi}_m$ has degree $2N$ and that Gauss-Legendre integration with $N+1$ nodes is exact for polynomials up to degree $2N+1$.

Scalar-valued Basis Functions on the Spatial Reference Element

For the vector-valued index set $\boldsymbol{\mathcal{N}} := \{0, 1, \dots, N\}^D$ let us define the set of scalar-valued spatial basis functions $\{\hat{\phi}_{\boldsymbol{n}}\}_{\boldsymbol{n} \in \boldsymbol{\mathcal{N}}}$ on $\hat{K} := [0,1]^D$ as

$$\hat{\phi}_{\boldsymbol{n}}(\boldsymbol{\xi}) = \prod_{d=1}^{D} \hat{\psi}_{[\boldsymbol{n}]_d}([\boldsymbol{\xi}]_d) = \hat{\psi}_{[\boldsymbol{n}]_d}([\boldsymbol{\xi}]_d), \tag{2.60}$$

i.e. $\{\hat{\phi}_{\boldsymbol{n}}\}_{\boldsymbol{n} \in \boldsymbol{\mathcal{N}}}$ is the tensor product of $\{\hat{\psi}_n\}_{n \in \mathcal{N}}$ and as such it is a basis of $\mathbb{Q}([0,1]^D) = \mathbb{Q}(\hat{K})$. If we define

$$\left[\hat{\boldsymbol{\xi}}_{\boldsymbol{n}}\right]_d = \hat{\xi}_{[\boldsymbol{n}]_d} \tag{2.61}$$

and

$$\prod_{d=1}^{D} \hat{\omega}_{[\boldsymbol{n}]_d}, \tag{2.62}$$

for all $d \in \mathcal{V}$ and $\boldsymbol{n} \in \boldsymbol{\mathcal{N}}$, we furthermore observe that the basis is orthogonal with respect to the L^2-scalar product, since

$$\begin{gathered}\left\langle \hat{\phi}_{\boldsymbol{n}}, \hat{\phi}_{\boldsymbol{m}} \right\rangle_{L^2\left(\hat{K}\right)} = \int_{\hat{K}} \hat{\phi}_{\boldsymbol{n}}(\boldsymbol{\xi})\hat{\phi}_{\boldsymbol{m}}(\boldsymbol{\xi})\, d\boldsymbol{\xi} = \\ \sum_{\boldsymbol{n}' \in \boldsymbol{\mathcal{N}}} \left(\hat{\omega}_{\boldsymbol{n}'} \hat{\phi}_{\boldsymbol{n}}(\hat{\boldsymbol{\xi}}_{\boldsymbol{n}'})\hat{\phi}_{\boldsymbol{m}}(\hat{\boldsymbol{\xi}}_{\boldsymbol{n}'}) \right) = \hat{\omega}_{\boldsymbol{n}} \delta_{\boldsymbol{nm}}\end{gathered} \tag{2.63}$$

for all $\boldsymbol{n}, \boldsymbol{m} \in \boldsymbol{\mathcal{N}}$. The natural extensions of the Kronecker delta for vector-valued indices is defined as follows:

$$\delta_{\boldsymbol{nm}} = \prod_{d=1}^{D} \delta_{[\boldsymbol{n}]_d [\boldsymbol{m}]_d} = \delta_{[\boldsymbol{n}]_d [\boldsymbol{m}]_d}. \tag{2.64}$$

Scalar-valued Basis Functions on the Space-time Reference Element

Analogously to the procedure illustrated above for the spatial reference element $\hat{K}$ we can define a basis $\{\hat{\theta}_{\boldsymbol{n}l}\}_{\boldsymbol{n}\in\boldsymbol{\mathcal{N}}, l\in\mathcal{N}}$ of $\mathbb{Q}_N(\hat{K} \times [0,1])$ on the reference space-time element $\hat{K} \times [0,1]$ as

$$\hat{\theta}_{\boldsymbol{n}l}(\boldsymbol{\xi}, \tau) = \hat{\phi}_{\boldsymbol{n}}(\boldsymbol{\xi}) \hat{\psi}_l(\tau), \tag{2.65}$$

which again is orthogonal, since

$$\left\langle \hat{\theta}_{\boldsymbol{n}l}, \hat{\theta}_{\boldsymbol{m}k} \right\rangle_{L^2\left(\hat{K}\times[0,1]\right)} = \int_0^1 \int_{\hat{K}} \hat{\theta}_{\boldsymbol{n}l} \hat{\theta}_{\boldsymbol{m}k} \, d\boldsymbol{\xi} d\tau = \hat{\omega}_{\boldsymbol{n}} \hat{\omega}_l \delta_{\boldsymbol{nm}} \delta_{lk} \tag{2.66}$$

for all $\boldsymbol{n}, \boldsymbol{m} \in \boldsymbol{\mathcal{N}}$ and $l, k \in \mathcal{N}$.

Vector-valued Basis Functions on the Spatial Reference Element

If we define $\{\hat{\boldsymbol{\phi}}_{\boldsymbol{n}v}\}_{\boldsymbol{n}\in\boldsymbol{\mathcal{N}}, v\in\mathcal{V}}$ as

$$\hat{\boldsymbol{\phi}}_{\boldsymbol{n}v} = \hat{\phi}_{\boldsymbol{n}} \boldsymbol{e}_v, \tag{2.67}$$

where $\boldsymbol{e}_v$ is the v-th unit vector, i.e.

$$[\boldsymbol{e}_v]_{v'} = \delta_{vv'} \tag{2.68}$$

for $v, v' \in \mathcal{V}$. Since

$$\left\langle \hat{\boldsymbol{\phi}}_{\boldsymbol{n}v}, \hat{\boldsymbol{\phi}}_{\boldsymbol{n}'v'} \right\rangle_{L^2\left(\hat{K}\right)^V} = \int_{\hat{K}} \left[\hat{\boldsymbol{\phi}}_{\boldsymbol{n}v}\right]_j \left[\hat{\boldsymbol{\phi}}_{\boldsymbol{n}'v'}\right]_j d\boldsymbol{\xi} =$$

$$\left([\boldsymbol{e}_v]_j \, [\boldsymbol{e}_{v'}]_j \right) \int_0^1 \int_{\hat{K}} \hat{\phi}_{\boldsymbol{n}} \hat{\phi}_{\boldsymbol{n}'} \, d\boldsymbol{\xi} = \hat{\omega}_{\boldsymbol{n}} \delta_{\boldsymbol{nn}'} \delta_{vv'} \tag{2.69}$$

for all $\boldsymbol{n}, \boldsymbol{n}' \in \boldsymbol{\mathcal{N}}$ and $v, v' \in \{1, 2, \ldots, V\}$ the set is an orthogonal basis for $\mathbb{Q}_N(\hat{K})^V$.

Vector-valued Basis Functions on the Space-time Reference Element

The set $\{\hat{\boldsymbol{\theta}}_{\boldsymbol{n}lv}\}_{\boldsymbol{n}\in\boldsymbol{\mathcal{N}}, l\in\mathcal{N}, v\in\mathcal{V}}$ defined as

$$\hat{\boldsymbol{\theta}}_{\boldsymbol{n}lv}(\boldsymbol{\xi}, \tau) = \hat{\theta}_{\boldsymbol{n}l}(\boldsymbol{\xi}, \tau) \boldsymbol{e}_v = \hat{\phi}_{\boldsymbol{n}}(\boldsymbol{\xi}) \hat{\psi}_l(\tau) \boldsymbol{e}_v \tag{2.70}$$

is a basis of $\mathbb{Q}_N(\hat{K} \times [0,1])^V$. Since furthermore

$$\left\langle \hat{\boldsymbol{\theta}}_{\boldsymbol{n}lv}, \hat{\boldsymbol{\theta}}_{\boldsymbol{n}'l'v'} \right\rangle_{L^2\left(\hat{K}\times[0,1]\right)^V} = \int_0^1 \int_{\hat{K}} \left[\hat{\boldsymbol{\theta}}_{\boldsymbol{n}lv}\right]_j \left[\hat{\boldsymbol{\theta}}_{\boldsymbol{n}'l'v'}\right]_j d\boldsymbol{\xi} d\tau = \hat{\omega}_{\boldsymbol{n}} \hat{\omega}_l \delta_{\boldsymbol{n}\boldsymbol{n}'} \delta_{ll'} \delta_{vv'}, \tag{2.71}$$

for all $\boldsymbol{n}, \boldsymbol{n}' \in \boldsymbol{\mathcal{N}}$, $l, l' \in \mathcal{N}$ and $v, v' \in \mathcal{V}$, the set is an orthogonal basis with respect to the respective L^2-scalar product.

2.1.10 Basis Functions in Global Coordinates

We can use the mappings derived in ch. 2.1.8 to map the basis functions to global coordinates. For the vector-valued basis functions on a spatial element K we obtain

$$\boldsymbol{\phi}^K_{\boldsymbol{n}v}(\boldsymbol{x}) = \begin{cases} \left(\hat{\boldsymbol{\phi}}_{\boldsymbol{n}v} \circ \boldsymbol{\mathcal{X}}_K^{-1}\right)(\boldsymbol{x}) & \text{if } \boldsymbol{x} \in K \\ 0 & \text{otherwise,} \end{cases} \tag{2.72}$$

and for the vector-valued basis functions on a space-time element $K \times [t_i, t_i + \Delta t_i]$ we have

$$\boldsymbol{\theta}^{Ki}_{\boldsymbol{n}lv}(\boldsymbol{x}, t) = \begin{cases} \left(\hat{\boldsymbol{\theta}}_{\boldsymbol{n}lv} \circ \left(\boldsymbol{\mathcal{X}}_K^{-1}, \mathcal{T}_i^{-1}\right)\right)(\boldsymbol{x}, t) & \text{if } \boldsymbol{x} \in K \text{ and } t \in [t_i, t_i + \Delta t_i] \\ 0 & \text{otherwise} \end{cases} \tag{2.73}$$

for $\boldsymbol{n} \in \boldsymbol{\mathcal{N}}$, $l \in \{0, 1, \ldots, N\}$ as well as $v \in \mathcal{V}$ and for all $K \in \mathcal{K}_h$ and $i \in \mathcal{I}$.

2.1.11 A Fully-discrete Iterative Method for the Space-time Predictor

We recall relation (2.38) for the space-time predictor. Plugging in the initial condition (2.37) yields

$$\begin{gathered} \int_K \left[\tilde{\boldsymbol{q}}_h^{K,i}\Big|_{t_i+\Delta t_i}\right]_j \left[\tilde{\boldsymbol{w}}_h^{K,i}\Big|_{t_i+\Delta t_i}\right]_j d\boldsymbol{x} - \int_{t_i}^{t_i+\Delta t_i}\int_K \left[\tilde{\boldsymbol{q}}_h^{K,i}\right]_j \frac{\partial}{\partial t}\left[\tilde{\boldsymbol{w}}_h^{K,i}\right]_j d\boldsymbol{x}dt = \\ \int_K \left[\tilde{\boldsymbol{u}}_h^{K,i}\Big|_{t_i}\right]_j \left[\tilde{\boldsymbol{w}}_h^{K,i}\Big|_{t_i}\right]_j d\boldsymbol{x} + \int_{t_i}^{t_i+\Delta t_i}\int_K \left[\boldsymbol{F}(\tilde{\boldsymbol{q}}_h^{K,i})\right]_{jk} \frac{\partial}{\partial x_k}\left[\tilde{\boldsymbol{w}}_h^{K,i}\right]_j d\boldsymbol{x}dt + \\ \int_{t_i}^{t_i+\Delta t_i}\int_K \left[\boldsymbol{s}\left(\tilde{\boldsymbol{q}}_h^{K,i}\right)\right]_j \left[\tilde{\boldsymbol{w}}_h^{K,i}\right]_j d\boldsymbol{x}dt, \end{gathered} \tag{2.74}$$

which we require to hold for all $\tilde{\boldsymbol{w}}_h \in \tilde{\mathbb{W}}_h$, $K \in \mathcal{K}_h$ and $i \in \mathcal{I}$.

Making use of the bases we derived in the previous section the cell-local space-time predictor $\tilde{\boldsymbol{q}}_h^{K,i}$ can be represented by a tensor of coefficients $\hat{\boldsymbol{q}}^{K,i}$ ("degrees of freedom") as follows:

$$\tilde{\boldsymbol{q}}_h^{K,i} = \left[\hat{\boldsymbol{q}}^{K,i}\right]_{\boldsymbol{n}lv} \boldsymbol{\theta}^{Ki}_{\boldsymbol{n}lv}. \tag{2.75}$$

The initial condition $\tilde{\boldsymbol{u}}_h^{K,i}\Big|_{t_i}$ can be represented as

$$\tilde{\boldsymbol{u}}_h^{K,i}\Big|_{t_i} = \left[\hat{\boldsymbol{u}}^{K,i}\right]_{\boldsymbol{n}v} \boldsymbol{\phi}^{K}_{\boldsymbol{n}v}, \tag{2.76}$$

where

$$\left[\hat{\boldsymbol{u}}^{K,i}\right]_{\boldsymbol{n}v} = \left[\tilde{\boldsymbol{u}}_h^{K,i}\Big|_{\left(\boldsymbol{\mathcal{X}}_K(\boldsymbol{\xi_n}),t_i\right)}\right]_v. \tag{2.77}$$

Inserting eqs. (2.75) and (2.76) into eq. (2.74) and introduction of the iteration index $r \in \{0, 1, \ldots, R\}$ leads to the following iterative scheme for the degrees of freedom of the cell-local space-time predictor:

$$\begin{aligned}
&\underbrace{\int_K \left[\left[\hat{\boldsymbol{q}}^{K,i,r+1}\right]_{\boldsymbol{n}lv} \boldsymbol{\theta}^{Ki}_{\boldsymbol{n}lv}\Big|_{t_i+\Delta t_i}\right]_j \left[\boldsymbol{\theta}^{Ki}_{\boldsymbol{\alpha}\beta\gamma}\Big|_{t_i+\Delta t_i}\right]_j d\boldsymbol{x}\, -}_{\text{S-I}} \\
&\underbrace{\int_{t_i}^{t_i+\Delta t_i}\int_K \left[\left[\hat{\boldsymbol{q}}^{K,i,r+1}\right]_{\boldsymbol{n}lv} \boldsymbol{\theta}^{Ki}_{\boldsymbol{n}lv}\right]_j \frac{\partial}{\partial t}\left[\boldsymbol{\theta}^{Ki}_{\boldsymbol{\alpha}\beta\gamma}\right]_j d\boldsymbol{x}dt =}_{\text{S-II}} \\
&\underbrace{\int_K \left[\left[\hat{\boldsymbol{u}}^{K,i}\right]_{\boldsymbol{n}v} \boldsymbol{\phi}^{K}_{\boldsymbol{n}v}\right]_j \left[\boldsymbol{\theta}^{Ki}_{\boldsymbol{\alpha}\beta\gamma}\Big|_{t_i}\right]_j d\boldsymbol{x}\, +}_{\text{S-III}} \\
&\underbrace{\int_{t_i}^{t_i+\Delta t_i}\int_K \left[\boldsymbol{F}\left(\left[\hat{\boldsymbol{q}}^{K,i,r}\right]_{\boldsymbol{n}lv} \boldsymbol{\theta}^{Ki}_{\boldsymbol{n}lv}\right)\right]_{jk} \frac{\partial}{\partial x_k}\left[\boldsymbol{\theta}^{Ki}_{\boldsymbol{\alpha}\beta\gamma}\right]_j d\boldsymbol{x}dt\, +}_{\text{S-IV}} \\
&\underbrace{\int_{t_i}^{t_i+\Delta t_i}\int_K \left[\boldsymbol{s}\left(\left[\hat{\boldsymbol{q}}^{K,i,r}\right]_{\boldsymbol{n}lv} \boldsymbol{\theta}^{Ki}_{\boldsymbol{n}lv}\right)\right]_j \left[\boldsymbol{\theta}^{Ki}_{\boldsymbol{\alpha}\beta\gamma}\right]_j d\boldsymbol{x}dt\,.}_{\text{S-V}}
\end{aligned} \tag{2.78}$$

We require this relation to hold for all $\boldsymbol{\alpha} \in \boldsymbol{\mathcal{N}}$, $\beta \in \mathcal{N}$ and $\gamma \in \mathcal{V}$.

As initial condition, i.e. for $r = 0$, we use

$$\left[\hat{\boldsymbol{q}}^{K,i,0}\right]_{\boldsymbol{n}vl} = \left[\hat{\boldsymbol{u}}^{K,i}\right]_{\boldsymbol{n}v} \tag{2.79}$$

for all time degrees of freedom $l \in \mathcal{N}$.

We will now proceed in a term-by-term fashion to rewrite all integrals with respect to reference coordinates so that we can finally derive a complete rule on how to compute $\hat{\boldsymbol{q}}^{K,i,r+1}$ that holds for all $K \in \mathcal{K}_h$ and $i \in \mathcal{I}$.

Term S-I

The first term of eq. (2.78) can be rewritten with respect to reference coordinates as follows:

$$
\begin{aligned}
&\int_K \left[\left[\hat{\boldsymbol{q}}^{K,i,r+1} \right]_{\boldsymbol{n}lw} \boldsymbol{\theta}^{Ki}_{\boldsymbol{n}lv} \Big|_{t_i+\Delta t_i} \right]_j \left[\boldsymbol{\theta}^{Ki}_{\boldsymbol{\alpha}\beta\gamma} \right]_{t_i+\Delta t_i} d\boldsymbol{x} = \\
&\int_K \left[\hat{\boldsymbol{q}}^{K,i,r+1} \right]_{\boldsymbol{n}lv} \phi^K_{\boldsymbol{n}} \left(\psi^i_l \Big|_{t_i+\Delta t_i} \right) [\boldsymbol{e}_v]_j \, \phi^K_{\boldsymbol{\alpha}} \left(\psi^i_\beta \Big|_{t_i+\Delta t_i} \right) \left[\boldsymbol{e}_\gamma \right]_j d\boldsymbol{x} = \\
&J\boldsymbol{\chi}_K \int_{\hat{K}} \left[\hat{\boldsymbol{q}}^{K,i,r+1} \right]_{\boldsymbol{n}lv} \hat{\phi}_{\boldsymbol{n}} \left(\hat{\psi}_l \Big|_1 \right) [\boldsymbol{e}_v]_j \, \hat{\phi}_{\boldsymbol{\alpha}} \left(\hat{\psi}_\beta \Big|_1 \right) \left[\boldsymbol{e}_\gamma \right]_j d\boldsymbol{\xi} = \\
&J\boldsymbol{\chi}_K \sum_{\boldsymbol{\alpha}' \in \boldsymbol{\mathcal{N}}} \left(\hat{\omega}_{\boldsymbol{\alpha}'} \left[\hat{\boldsymbol{q}}^{K,i,r+1} \right]_{\boldsymbol{n}lv} \hat{\phi}_{\boldsymbol{n}}(\hat{\boldsymbol{\xi}}_{\boldsymbol{\alpha}'}) \left(\hat{\psi}_l \Big|_1 \right) [\boldsymbol{e}_v]_j \, \hat{\phi}_{\boldsymbol{\alpha}}(\hat{\boldsymbol{\xi}}_{\boldsymbol{\alpha}'}) \left(\hat{\psi}_\beta \Big|_1 \right) \left[\boldsymbol{e}_\gamma \right]_j \right) = \\
&J\boldsymbol{\chi}_K \sum_{\boldsymbol{\alpha}' \in \boldsymbol{\mathcal{N}}} \left(\hat{\omega}_{\boldsymbol{\alpha}'} \left[\hat{\boldsymbol{q}}^{K,i,r+1} \right]_{\boldsymbol{n}lv} \delta_{\boldsymbol{n}\boldsymbol{\alpha}'} \left(\hat{\psi}_l \Big|_1 \right) \delta_{vj} \delta_{\boldsymbol{\alpha}\boldsymbol{\alpha}'} \left(\hat{\psi}_\beta \Big|_1 \right) \delta_{j\gamma} \right) = \\
&J\boldsymbol{\chi}_K \hat{\omega}_{\boldsymbol{\alpha}} \left[\hat{\psi}_\beta \Big|_1 \hat{\psi}_l \Big|_1 \right] \left[\hat{\boldsymbol{q}}^{K,i,r+1} \right]_{\boldsymbol{\alpha}l\gamma} = \\
&J\boldsymbol{\chi}_K \hat{\omega}_{\boldsymbol{\alpha}} \left[\boldsymbol{R} \right]_{\beta,l} \left[\hat{\boldsymbol{q}}^{K,i,r+1} \right]_{\boldsymbol{\alpha}l\gamma},
\end{aligned}
\tag{2.80}
$$

where we remember from eq. (2.48) that

$$
J\boldsymbol{\chi}_K = \prod_{d=1}^{D} [\Delta \boldsymbol{x}]_d \tag{2.81}
$$

and we define the matrix $\boldsymbol{R}$ representing the Right Reference Element Mass Operator as

$$
[\boldsymbol{R}]_{i,j} := \left[\hat{\psi}_i \Big|_1 \hat{\psi}_j \Big|_1 \right]_{i,j} \tag{2.82}
$$

for $i, j \in \mathcal{N}$. A Python script to compute $\boldsymbol{R}$ can be found in appendix A.

Term S-II

The second term of eq. (2.78) can be rewritten with respect to reference coordinates as follows:

$$
\begin{aligned}
&\int_{t_i}^{t_i+\Delta t_i}\int_K \left[\left[\hat{\boldsymbol{q}}^{K,i,r+1}\right]_{\boldsymbol{n}l v} \boldsymbol{\theta}^{Ki}_{\boldsymbol{n}lv}\right]_j \frac{\partial}{\partial t}\left[\boldsymbol{\theta}^{Ki}_{\boldsymbol{\alpha}\beta\gamma}\right]_j \, d\boldsymbol{x}dt = \\
&\int_{t_i}^{t_i+\Delta t_i}\int_K \left[\hat{\boldsymbol{q}}^{K,i,r+1}\right]_{\boldsymbol{n}lv} \phi^K_{\boldsymbol{n}} \psi^i_l \left[\boldsymbol{e}_v\right]_j \phi^K_{\boldsymbol{\alpha}} \left(\frac{\partial}{\partial t}\psi^i_\beta\right)\left[\boldsymbol{e}_\gamma\right]_j \, d\boldsymbol{x}dt = \\
&J_{\mathcal{T}_i} J_{\boldsymbol{\mathcal{X}}_K} \int_0^1 \int_{\hat{K}} \left[\hat{\boldsymbol{q}}^{K,i,r+1}\right]_{\boldsymbol{n}lv} \hat{\phi}_{\boldsymbol{n}} \hat{\psi}_l \left[\boldsymbol{e}_v\right]_j \hat{\phi}_{\boldsymbol{\alpha}} \left(\frac{1}{\Delta t_i}\frac{\partial}{\partial \tau}\hat{\psi}_\beta\right)\left[\boldsymbol{e}_\gamma\right]_j \, d\boldsymbol{\xi}d\tau = \\
&J_{\mathcal{T}_i} J_{\boldsymbol{\mathcal{X}}_K} \sum_{\boldsymbol{\alpha}'\in\boldsymbol{\mathcal{N}}} \sum_{\beta'\in\mathcal{N}} \Bigg(\hat{\omega}_{\boldsymbol{\alpha}'}\hat{\omega}_{\beta'} \left[\hat{\boldsymbol{q}}^{K,i,r+1}\right]_{\boldsymbol{n}lv} \hat{\phi}_{\boldsymbol{n}}(\hat{\boldsymbol{\xi}}_{\boldsymbol{\alpha}'})\hat{\psi}_l(\hat{\tau}_{\beta'}) \left[\boldsymbol{e}_v\right]_j \dots \\
&\qquad\qquad \dots \hat{\phi}_{\boldsymbol{\alpha}}(\hat{\boldsymbol{\xi}}_{\boldsymbol{\alpha}'}) \left(\frac{\partial}{\partial\tau}\hat{\psi}_\beta(\hat{\tau}_{\beta'})\right)\left[\boldsymbol{e}_\gamma\right]_j \Bigg) = \\
&J_{\mathcal{T}_i} J_{\boldsymbol{\mathcal{X}}_K} \sum_{\boldsymbol{\alpha}'\in\boldsymbol{\mathcal{N}}} \sum_{\beta'\in\mathcal{N}} \Bigg(\hat{\omega}_{\boldsymbol{\alpha}'}\hat{\omega}_{\beta'} \left[\hat{\boldsymbol{q}}^{K,i,r+1}\right]_{\boldsymbol{n}lv} \delta_{\boldsymbol{n}\boldsymbol{\alpha}'}\delta_{l\beta'}\delta_{vj} \dots \\
&\qquad\qquad \dots \delta_{\boldsymbol{\alpha}\boldsymbol{\alpha}'} \left(\frac{1}{\Delta t_i}\frac{\partial}{\partial\tau}\hat{\psi}_\beta(\hat{\tau}_{\beta'})\right)\delta_{\gamma j} \Bigg) = \\
&J_{\mathcal{T}_i} J_{\boldsymbol{\mathcal{X}}_K} \hat{\omega}_{\boldsymbol{\alpha}} \frac{1}{\Delta t_i} \sum_{\beta'\in\mathcal{N}} \left(\hat{\omega}_{\beta'} \left[\frac{\partial}{\partial\tau}\hat{\psi}_\beta(\hat{\tau}_{\beta'})\right] \left[\hat{\boldsymbol{q}}^{K,i,r+1}\right]_{\boldsymbol{\alpha}\beta'\gamma} \right) = \\
&J_{\mathcal{T}_i} J_{\boldsymbol{\mathcal{X}}_K} \hat{\omega}_{\boldsymbol{\alpha}} \frac{1}{\Delta t_i} \left[\boldsymbol{K}\right]_{\beta,\beta'} \left[\hat{\boldsymbol{q}}^{K,i,r+1}\right]_{\boldsymbol{\alpha}\beta'\gamma}
\end{aligned}
\tag{2.83}
$$

where we remember from eq. (2.50) that

$$J_{\mathcal{T}_i} = \Delta t_i, \tag{2.84}$$

so that Δt_i and $1/\Delta t_i$ in eq. (2.83) cancel. In the derivation we made use of the fact that due to the chain rule

$$\frac{\partial}{\partial t}\psi^i_\beta = \frac{\partial}{\partial t}\left(\hat{\psi}_\beta \circ \mathcal{T}_i^{-1}\right) = \left(\frac{\partial}{\partial\tau}\hat{\psi}_\beta\right)\left(\frac{\partial}{\partial t}\mathcal{T}_i^{-1}\right) = \frac{1}{\Delta t_i}\frac{\partial}{\partial\tau}\hat{\psi}_\beta. \tag{2.85}$$

We furthermore introduce the matrix $\boldsymbol{K}$ representing the Reference Element Stiffness Operator given as

$$[\boldsymbol{K}]_{ij} = \hat{\omega}_j \frac{\partial}{\partial\tau}\hat{\psi}_i(\hat{\tau}_j) \tag{2.86}$$

for $i, j \in \mathcal{N}$. A Python script to compute $\boldsymbol{K}$ can be found in appendix A.

Term S-III

The third term of eq. (2.78) can be rewritten with respect to reference coordinates as follows:

$$
\begin{aligned}
&\int_K \left[\left[\hat{\boldsymbol{u}}^{K,i} \right]_{\boldsymbol{n}v} \boldsymbol{\phi}^K_{\boldsymbol{n}v} \right]_j \left[\boldsymbol{\theta}^{Ki}_{\boldsymbol{\alpha}\beta\gamma} \Big|_{t_i} \right]_j d\boldsymbol{x} = \\
&\int_K \left[\hat{\boldsymbol{u}}^{K,i} \right]_{\boldsymbol{n}v} \phi^K_{\boldsymbol{n}} \left[\boldsymbol{e}_v \right]_j \phi^K_{\boldsymbol{\alpha}} \left(\psi^i_\beta \Big|_{t_i} \right) \left[\boldsymbol{e}_\gamma \right]_j d\boldsymbol{x} = \\
&J\boldsymbol{\chi}_K \int_{\hat{K}} \left[\hat{\boldsymbol{u}}^{K,i} \right]_{\boldsymbol{n}v} \hat{\phi}_{\boldsymbol{n}} \left[\boldsymbol{e}_v \right]_j \hat{\phi}_{\boldsymbol{\alpha}} \left(\hat{\psi}_\beta \Big|_0 \right) \left[\boldsymbol{e}_\gamma \right]_j d\boldsymbol{\xi} = \\
&J\boldsymbol{\chi}_K \sum_{\boldsymbol{\alpha}' \in \boldsymbol{\mathcal{N}}} \left(\hat{\omega}_{\boldsymbol{\alpha}'} \left[\hat{\boldsymbol{u}}^{K,i} \right]_{\boldsymbol{n}v} \hat{\phi}_{\boldsymbol{n}}(\boldsymbol{\xi}_{\boldsymbol{\alpha}'}) \left[\boldsymbol{e}_v \right]_j \hat{\phi}_{\boldsymbol{\alpha}}(\boldsymbol{\xi}_{\boldsymbol{\alpha}'}) \left(\hat{\psi}_\beta \Big|_0 \right) \left[\boldsymbol{e}_\gamma \right]_j \right) = \\
&J\boldsymbol{\chi}_K \sum_{\boldsymbol{\alpha}' \in \boldsymbol{\mathcal{N}}} \left(\hat{\omega}_{\boldsymbol{\alpha}'} \left[\hat{\boldsymbol{u}}^{K,i} \right]_{\boldsymbol{n}v} \delta_{\boldsymbol{n}\boldsymbol{\alpha}'} \delta_{vj} \delta_{\boldsymbol{\alpha}\boldsymbol{\alpha}'} \left(\hat{\psi}_\beta \Big|_0 \right) \delta_{\gamma j} \right) = \\
&J\boldsymbol{\chi}_K \hat{\omega}_{\boldsymbol{\alpha}} \left[\hat{\psi}_\beta \Big|_0 \right] \left[\hat{\boldsymbol{u}}^{K,i} \right]_{\boldsymbol{\alpha}\gamma} = \\
&J\boldsymbol{\chi}_K \hat{\omega}_{\boldsymbol{\alpha}} \left[\boldsymbol{l} \right]_\beta \left[\hat{\boldsymbol{u}}^{K,i} \right]_{\boldsymbol{\alpha}\gamma},
\end{aligned}
\tag{2.87}
$$

where we define the vector $\boldsymbol{l}$ representing the Left Reference Element Flux Operator as

$$
\left[\boldsymbol{l} \right]_i = \hat{\psi}_i \Big|_0 \tag{2.88}
$$

for $i \in \mathcal{N}$. A Python script to compute $\boldsymbol{l}$ can be found in appendix A.

Term S-IV

The third term of eq. (2.78) can be rewritten with respect to reference coordinates as follows:

$$
\begin{aligned}
&\int_{t_i}^{t_i+\Delta t_i}\int_K \left[\boldsymbol{F}\left(\left[\hat{\boldsymbol{q}}^{K,i,r}\right]_{nlv}\boldsymbol{\theta}^{Ki}_{\boldsymbol{n}lv}\right)\right]_{jk}\frac{\partial}{\partial x_k}\left[\boldsymbol{\theta}^{Ki}_{\boldsymbol{\alpha}\beta\gamma}\right]_j\, d\boldsymbol{x}dt = \\
&\int_{t_i}^{t_i+\Delta t_i}\int_K \left[\boldsymbol{F}\left(\left[\hat{\boldsymbol{q}}^{K,i,r}\right]_{nlv}\phi^K_{\boldsymbol{n}}\psi^i_l\boldsymbol{e}_v\right)\right]_{jk}\left(\prod_{d=1,d\neq k}^{D}\psi^K_{[\boldsymbol{\alpha}]_d}([\boldsymbol{x}]_d)\right)\psi^i_\beta(t)\left[\boldsymbol{e}_\gamma\right]_j\cdots \\
&\cdots\left(\frac{\partial}{\partial x_k}\psi^K_{[\boldsymbol{\alpha}]_k}\right)d\boldsymbol{x}dt = \\
&J_{\mathcal{T}_i}J_{\boldsymbol{\mathcal{X}}_K}\int_0^1\int_{\hat{K}}\left[\boldsymbol{F}\left(\left[\hat{\boldsymbol{q}}^{K,i,r}\right]_{nlv}\hat{\phi}_{\boldsymbol{n}}\hat{\psi}_l\boldsymbol{e}_v\right)\right]_{jk}\left(\prod_{d=1,d\neq k}^{D}\hat{\psi}_{[\boldsymbol{\alpha}]_d}([\boldsymbol{\xi}]_d)\right)\hat{\psi}_\beta(t)\left[\boldsymbol{e}_\gamma\right]_j\cdots \\
&\cdots\left(\frac{1}{[\Delta\boldsymbol{x}]_k}\frac{\partial}{\partial\xi_k}\hat{\psi}_{[\boldsymbol{\alpha}]_k}([\boldsymbol{\xi}]_k)\right)d\boldsymbol{\xi}d\tau = \\
&J_{\mathcal{T}_i}J_{\boldsymbol{\mathcal{X}}_K}\sum_{\boldsymbol{\alpha}'\in\boldsymbol{\mathcal{N}}}\sum_{\beta'\in\mathcal{N}}\left(\hat{\omega}_{\boldsymbol{\alpha}'}\hat{\omega}_{\beta'}\left[\boldsymbol{F}\left(\left[\hat{\boldsymbol{q}}^{K,i,r}\right]_{nlv}\hat{\phi}_{\boldsymbol{n}}(\hat{\boldsymbol{\xi}}_{\boldsymbol{\alpha}'})\hat{\psi}_l(\hat{\tau}_{\beta'})\boldsymbol{e}_v\right)\right]_{jk}\cdots\right. \\
&\left.\cdots\left(\prod_{d=1,d\neq k}^{D}\hat{\psi}_{[\boldsymbol{\alpha}]_d}\left(\left[\hat{\boldsymbol{\xi}}_{\boldsymbol{\alpha}'}\right]_d\right)\right)\hat{\psi}_\beta(\hat{\tau}_{\beta'})\left[\boldsymbol{e}_\gamma\right]_j\left(\frac{1}{[\Delta\boldsymbol{x}]_k}\frac{\partial}{\partial\xi_k}\hat{\psi}_{[\boldsymbol{\alpha}]_k}\left(\left[\hat{\boldsymbol{\xi}}_{\boldsymbol{\alpha}'}\right]_k\right)\right)\right) = \\
&J_{\mathcal{T}_i}J_{\boldsymbol{\mathcal{X}}_K}\sum_{\boldsymbol{\alpha}'\in\boldsymbol{\mathcal{N}}}\sum_{\beta'\in\mathcal{N}}\left(\hat{\omega}_{\boldsymbol{\alpha}'}\hat{\omega}_{\beta'}\left[\boldsymbol{F}\left(\left[\hat{\boldsymbol{q}}^{K,i,r}\right]_{nlv}\delta_{\boldsymbol{n}\boldsymbol{\alpha}'}\delta_{l\beta'}\boldsymbol{e}_v\right)\right]_{jk}\cdots\right. \\
&\left.\cdots\left(\prod_{d=1,d\neq k}^{D}\delta_{[\boldsymbol{\alpha}]_d[\boldsymbol{\alpha}']_d}\right)\delta_{\beta\beta'}\delta_{\gamma j}\left(\frac{1}{[\Delta\boldsymbol{x}]_k}\frac{\partial}{\partial\xi_k}\hat{\psi}_{[\boldsymbol{\alpha}]_k}\left(\left[\hat{\boldsymbol{\xi}}_{\boldsymbol{\alpha}'}\right]_k\right)\right)\right) = \\
&J_{\mathcal{T}_i}J_{\boldsymbol{\mathcal{X}}_K}\hat{\omega}_\beta\sum_{k=1}^{D}\left(\frac{1}{[\Delta\boldsymbol{x}]_k}\sum_{\alpha'_k\in\{0,1,\dots,N\}}\left(\prod_{d=0,d\neq k}^{D}\hat{\omega}_{[\boldsymbol{\alpha}]_d}\cdots\right.\right. \\
&\left.\left.\cdots\hat{\omega}_{\alpha'_k}\left(\frac{\partial}{\partial\xi_k}\hat{\psi}_{[\boldsymbol{\alpha}]_k}\left(\hat{\xi}_{\alpha'_k}\right)\right)\left[\boldsymbol{F}\left(\left[\hat{\boldsymbol{q}}^{K,i,r}\right]_{\left[[\boldsymbol{\alpha}]_{1:k-1},\alpha'_k,[\boldsymbol{\alpha}]_{k+1:N}\right]}\right)\right]_{jk}\right)\right) = \\
&J_{\mathcal{T}_i}J_{\boldsymbol{\mathcal{X}}_K}\hat{\omega}_\beta\sum_{k=1}^{D}\left(\frac{1}{[\Delta\boldsymbol{x}]_k}\sum_{\alpha'_k\in\{0,1,\dots,N\}}\left(\prod_{d=0,d\neq k}^{D}\hat{\omega}_{[\boldsymbol{\alpha}]_d}\cdots\right.\right. \\
&\left.\left.\cdots[\boldsymbol{K}]_{[\boldsymbol{\alpha}]_k,\alpha'_k}\left[\boldsymbol{F}\left(\left[\hat{\boldsymbol{q}}^{K,i,r}\right]_{[\alpha_0,\alpha_1,\dots,\alpha_{k-1},\alpha'_k,\alpha_{k+1},\dots,\alpha_N]\beta v}\boldsymbol{e}_v\right)\right]_{\gamma k}\right)\right),
\end{aligned}
\tag{2.89}
$$

where we used that

$$
\begin{aligned}
&\frac{\partial}{\partial x_k}\boldsymbol{\theta}^{Ki}_{\boldsymbol{\alpha}\beta\gamma}(\boldsymbol{x},t) = \left(\frac{\partial}{\partial x_k}\phi^K_{\boldsymbol{\alpha}}(\boldsymbol{x})\right)\psi^i_\beta(t)\boldsymbol{e}_\gamma = \left(\frac{\partial}{\partial x_k}\prod_{d=1}^{D}\psi^K_{[\boldsymbol{\alpha}]_d}([\boldsymbol{x}]_d)\right)\psi^i_\beta(t)\boldsymbol{e}_\gamma = \\
&\left(\prod_{d=1,d\neq k}^{D}\psi^K_{[\boldsymbol{\alpha}]_d}([\boldsymbol{x}]_d)\right)\left(\frac{\partial}{\partial x_k}\psi^K_{[\boldsymbol{\alpha}]_k}([\boldsymbol{x}]_k)\right)\psi^i_\beta(t)\boldsymbol{e}_\gamma = \\
&\left(\prod_{d=1,d\neq k}^{D}\psi^K_{[\boldsymbol{\alpha}]_d}([\boldsymbol{x}]_d)\right)\left(\frac{\partial}{\partial x_k}\hat{\psi}_{[\boldsymbol{\alpha}]_k}\left(\left[\boldsymbol{\mathcal{X}}^{-1}_K(\boldsymbol{x})\right]_k\right)\right)\psi^i_\beta(t)\boldsymbol{e}_\gamma = \\
&\left(\prod_{d=1,d\neq k}^{D}\psi^K_{[\boldsymbol{\alpha}]_d}([\boldsymbol{x}]_d)\right)\left(\left(\frac{\partial}{\partial \xi_k}\hat{\psi}_{[\boldsymbol{\alpha}]_k}\left(\left[\boldsymbol{\mathcal{X}}^{-1}_K(\boldsymbol{x})\right]_k\right)\right)\left(\frac{\partial}{\partial x_k}\left[\boldsymbol{\mathcal{X}}^{-1}_K(\boldsymbol{x})\right]_k\right)\right)\dots \\
&\dots\psi^i_\beta(t)\boldsymbol{e}_\gamma = \\
&\left(\prod_{d=1,d\neq k}^{D}\psi^K_{[\boldsymbol{\alpha}]_d}([\boldsymbol{x}]_d)\right)\left(\frac{1}{\left[\Delta\boldsymbol{x}^K\right]_k}\frac{\partial}{\partial \xi_k}\hat{\phi}_{[\boldsymbol{\alpha}]_k}\left(\left[\boldsymbol{\mathcal{X}}^{-1}_K(\boldsymbol{x})\right]_k\right)\right)\psi^i_\beta(t)\boldsymbol{e}_\gamma.
\end{aligned}
\tag{2.90}
$$

Term S-V

The fifth term of eq. (2.78) can be rewritten with respect to reference coordinates as follows:

$$
\begin{aligned}
&\int_{t_i}^{t_i+\Delta t_i}\int_K\left[\boldsymbol{s}\left(\left[\hat{\boldsymbol{q}}^{K,i,r}\right]_{nlv}\boldsymbol{\theta}^{Ki}_{nlv}\right)\right]_j\left[\boldsymbol{\theta}^{Ki}_{\boldsymbol{\alpha}\beta\gamma}\right]_j d\boldsymbol{x}dt = \\
&J_{\mathcal{T}_i}J_{\boldsymbol{\mathcal{X}}_K}\int_0^1\int_{\hat{K}}\left[\boldsymbol{s}\left(\left[\hat{\boldsymbol{q}}^{K,i,r}\right]_{nlv}\hat{\phi}_{\boldsymbol{n}}\hat{\psi}_l\boldsymbol{e}_v\right)\right]_j\hat{\phi}_{\boldsymbol{\alpha}}\hat{\psi}_l\left[\boldsymbol{e}_\gamma\right]_j d\boldsymbol{\xi}d\tau = \\
&J_{\mathcal{T}_i}J_{\boldsymbol{\mathcal{X}}_K}\sum_{\boldsymbol{\alpha}'\in\boldsymbol{\mathcal{N}}}\sum_{\beta'\in\mathcal{N}}\left(\hat{\omega}_{\boldsymbol{\alpha}'}\hat{\omega}_{\beta'}\left[\boldsymbol{s}\left(\left[\hat{\boldsymbol{q}}^{K,i,r}\right]_{nlv}\hat{\phi}_{\boldsymbol{n}}(\boldsymbol{\xi}_{\boldsymbol{\alpha}'})\hat{\psi}_l(\hat{\tau}_{\beta'})\boldsymbol{e}_v\right)\right]_j\dots\right. \\
&\left.\dots\hat{\phi}_{\boldsymbol{\alpha}}(\boldsymbol{\xi}_{\boldsymbol{\alpha}'})\hat{\psi}_\beta(\hat{\tau}_{\beta'})\left[\boldsymbol{e}_\gamma\right]_j\right) = \\
&J_{\mathcal{T}_i}J_{\boldsymbol{\mathcal{X}}_K}\sum_{\boldsymbol{\alpha}'\in\boldsymbol{\mathcal{N}}}\sum_{\beta'\in\mathcal{N}}\left(\hat{\omega}_{\boldsymbol{\alpha}'}\hat{\omega}_{\beta'}\left[\boldsymbol{s}\left(\left[\hat{\boldsymbol{q}}^{K,i,r}\right]_{nlv}\delta_{\boldsymbol{n}\boldsymbol{\alpha}'}\delta_{l\beta'}\boldsymbol{e}_v\right)\right]_j\delta_{\boldsymbol{\alpha}\boldsymbol{\alpha}'}\delta_{\beta\beta'}\delta_{\gamma j}\right) = \\
&J_{\mathcal{T}_i}J_{\boldsymbol{\mathcal{X}}_K}\hat{\omega}_{\boldsymbol{\alpha}}\hat{\omega}_\beta\left[\boldsymbol{s}\left(\left[\hat{\boldsymbol{q}}^{K,i,r}\right]_{\boldsymbol{\alpha}\beta v}\boldsymbol{e}_v\right)\right]_\gamma
\end{aligned}
\tag{2.91}
$$

The Complete Fixed-point Iteration

Now collecting the results from eqs. (2.80), (2.83), (2.87), (2.89) and (2.91) and plugging them back into eq. (2.78) and division by $J_{\mathcal{X}_K}$ and $\hat{\omega}_{\boldsymbol{\alpha}}$ yields

$$\begin{aligned}
&(\boldsymbol{R}-\boldsymbol{K})_{\beta\beta'}\left[\hat{\boldsymbol{q}}^{K,i,r+1}\right]_{\boldsymbol{\alpha}\beta'\gamma} = [\boldsymbol{l}]_{\beta}\left[\hat{\boldsymbol{u}}^{K,i}\right]_{\boldsymbol{\alpha}\gamma} + \\
&J_{\mathcal{T}_i}\frac{\hat{\omega}_{\beta}}{\hat{\omega}_{\boldsymbol{\alpha}}}\sum_{k=1}^{D}\left(\frac{1}{[\Delta\boldsymbol{x}]_k}\sum_{\alpha'_k\in\{0,1,\ldots,N\}}\left(\prod_{d=0,d\neq k}^{D}\hat{\omega}_{[\boldsymbol{\alpha}]_d}\cdots\right.\right. \\
&\left.\left.\ldots[\boldsymbol{K}]_{[\boldsymbol{\alpha}]_k,\alpha'_k}\left[\boldsymbol{F}\left(\left[\hat{\boldsymbol{q}}^{K,i,r}\right]_{[\alpha_0,\alpha_1,\ldots,\alpha_{k-1},\alpha'_k,\alpha_{k+1},\ldots,\alpha_N]\beta v}\boldsymbol{e}_v\right)\right]_{\gamma k}\right)\right) + \\
&J_{\mathcal{T}_i}\hat{\omega}_{\beta}\left[\boldsymbol{s}\left(\left[\hat{\boldsymbol{q}}^{K,i,r}\right]_{\boldsymbol{\alpha}\beta v}\boldsymbol{e}_v\right)\right]_{\gamma},
\end{aligned} \tag{2.92}$$

which has to hold for all $\boldsymbol{\alpha}\in\boldsymbol{\mathcal{N}}$, $\beta\in\mathcal{N}$ and $\gamma\in\mathcal{V}$. To speed up the computation of the new iterate $\hat{\boldsymbol{q}}^{K,i,r+1}$ we can invert the matrix on the left-hand side prior to the simulation to obtain the iteration matrix $\tilde{\boldsymbol{K}} := (\boldsymbol{R}-\boldsymbol{K})^{-1}$. A Python script to compute $\tilde{\boldsymbol{K}}$ can be found in appendix A.

A possible termination criterion could be $\Delta < \varepsilon$, where $\varepsilon > 0$ is a suitable constant related to the desired accuracy of the iteration, e.g. $\varepsilon = 10^{-7}$ and the squared element-wise residual Δ^2 is defined as follows:

$$\Delta^2 = \sum_{\boldsymbol{n}\in\boldsymbol{\mathcal{N}}}\sum_{l\in\mathcal{N}}\sum_{v\in\mathcal{V}}\left(\left[\hat{\boldsymbol{q}}^{K,i,r+1}\right]_{\boldsymbol{n},l,v} - \left[\hat{\boldsymbol{q}}^{K,i,r}\right]_{\boldsymbol{n},l,v}\right). \tag{2.93}$$

For linear homogeneous scalar hyperbolic balance laws and neglecting floating point errors it can be proven that the iteration converges after at most N steps (see **??** for details).

2.1.12 A Fully-discrete Update Scheme for the Time-discrete Solution

Now that we have developed a method to compute the space-time predictor, we can go back to the original one-step, cell-local update scheme given in eq. (2.34). Inserting the local space-time predictor $\tilde{\boldsymbol{q}}_h^{K,i}$ yields

$$\begin{aligned}
&\int_K \left[\tilde{\boldsymbol{u}}_h^{K,i}\Big|_{t_i+\Delta t_i} \right]_v \left[\boldsymbol{w}_h^K\right]_v d\boldsymbol{x} = \int_K \left[\tilde{\boldsymbol{u}}_h^{K,i}\Big|_{t_i} \right]_v \left[\boldsymbol{w}_h^K\right]_v d\boldsymbol{x} + \\
&\int_{t_i}^{t_i+\Delta t_i} \int_K \left[\boldsymbol{F}(\tilde{\boldsymbol{q}}_h^{K,i})\right]_{vd} \frac{\partial}{\partial x_d} \left[\boldsymbol{w}_h^K\right]_v d\boldsymbol{x}dt + \\
&\int_{t_i}^{t_i+\Delta t_i} \int_K \left[\boldsymbol{s}\left(\hat{\boldsymbol{q}}^{K,i}\right)\right]_v \left[\boldsymbol{w}_h^K\right]_v d\boldsymbol{x}dt - \\
&\int_{t_i}^{t_i+\Delta t_i} \int_{\partial K} \left[\boldsymbol{\mathcal{G}}\left(\tilde{\boldsymbol{q}}_h^{K,i}, \tilde{\boldsymbol{q}}_h^{K^+i}, \boldsymbol{n}\right)\right]_v \left[\boldsymbol{w}_h^K\right]_v ds(\boldsymbol{x})dt,
\end{aligned} \tag{2.94}$$

which has to hold for all $v \in \mathcal{V}$, $K \in \mathcal{K}_h$, $\boldsymbol{w}_h \in \mathbb{W}_h$ and $i \in \mathcal{I}$.

Making use of the bases we derived earlier the call-local solution $\tilde{\boldsymbol{u}}_h^{K,i}$ at times t_i and $t_i + \Delta t_i$ can be represented by tensors of coefficients $\hat{\boldsymbol{u}}^{K,i}$ and $\hat{\boldsymbol{u}}^{K,i+1}$ as

$$\tilde{\boldsymbol{u}}_h^{K,i}\Big|_{t_i} = \left[\hat{\boldsymbol{u}}^{K,i}\right]_{\boldsymbol{n},v} \boldsymbol{\phi}_{\boldsymbol{n},v}^K \tag{2.95}$$

and

$$\tilde{\boldsymbol{u}}_h^{K,i}\Big|_{t_i+\Delta t_i} = \left[\hat{\boldsymbol{u}}^{K,i+1}\right]_{\boldsymbol{n},v} \boldsymbol{\phi}_{\boldsymbol{n},v}^K, \tag{2.96}$$

respectively. Inserting eqs. (2.95) and (2.96) and the ansatz for the space-time predictor (2.75) into eq. (2.94) yields

$$\begin{aligned}
&\underbrace{\int_K \left[\left[\hat{\boldsymbol{u}}^{K,i+1}\right]_{\boldsymbol{n},v} \boldsymbol{\phi}_{\boldsymbol{n},v}^K\right]_j \left[\boldsymbol{\phi}_{\boldsymbol{\alpha},\gamma}^K\right]_j d\boldsymbol{x}}_{\text{U-I}} = \underbrace{\int_K \left[\left[\hat{\boldsymbol{u}}^{K,i}\right]_{\boldsymbol{n},v} \boldsymbol{\phi}_{\boldsymbol{n},v}^K\right]_j \left[\boldsymbol{\phi}_{\boldsymbol{\alpha},\gamma}^K\right]_j}_{\text{U-II}} + \\
&\underbrace{\int_{t_i}^{t_i+\Delta t_i} \int_K \left[\boldsymbol{F}\left(\left[\hat{\boldsymbol{q}}^{K,i}\right]_{\boldsymbol{n},l,v} \boldsymbol{\theta}_{\boldsymbol{n},l,v}^{Ki}\right)\right]_{jk} \frac{\partial}{\partial x_k} \left[\boldsymbol{\phi}_{\boldsymbol{\alpha},\gamma}^K\right]_j d\boldsymbol{x}dt +}_{\text{U-III}} \\
&\underbrace{\int_{t_i}^{t_i+\Delta t_i} \int_K \left[\boldsymbol{s}\left(\left[\hat{\boldsymbol{q}}^{K,i}\right]_{\boldsymbol{n},l,v} \boldsymbol{\theta}_{\boldsymbol{n},l,v}^{Ki}\right)\right]_j \left[\boldsymbol{\phi}_{\boldsymbol{\alpha},\gamma}^K\right]_j d\boldsymbol{x}dt -}_{\text{U-IV}} \\
&\underbrace{\int_{t_i}^{t_i+\Delta t_i} \int_{\partial K} \left[\boldsymbol{\mathcal{G}}\left(\hat{\boldsymbol{q}}^{K,i}, \hat{\boldsymbol{q}}^{K^+,i}, \boldsymbol{n}\right)\right]_j \left[\boldsymbol{\phi}_{\boldsymbol{\alpha},\gamma}^K\right]_j ds(\boldsymbol{x})dt,}_{\text{U-V}}
\end{aligned} \tag{2.97}$$

which we require to hold for all $\boldsymbol{\alpha} \in \boldsymbol{\mathcal{N}}$, $\gamma \in \mathcal{V}$, $K \in \mathcal{K}_h$ and $i \in \mathcal{I}$. In the following we will again proceed by simplifying each term in reference coordinates separately and then in the end assemble all terms to obtain a complete fully-discrete update scheme.

Term U-I

The first term of eq. (2.97) can be rewritten with respect to reference coordinates as follows:

$$
\begin{aligned}
&\int_K \left[\left[\hat{\boldsymbol{u}}^{K,i+1}\right]_{\boldsymbol{n},v} \boldsymbol{\phi}^K_{\boldsymbol{n},v} \right]_j \left[\boldsymbol{\phi}^K_{\boldsymbol{\alpha},\gamma}\right]_j \, d\boldsymbol{x} = \\
&\int_K \left[\left[\hat{\boldsymbol{u}}^{K,i+1}\right]_{\boldsymbol{n},v} \phi^K_{\boldsymbol{n}} \boldsymbol{e}_v \right]_j \left[\phi^K_{\boldsymbol{\alpha}} \boldsymbol{e}_\gamma\right]_j \, d\boldsymbol{x} = \\
&J_{\boldsymbol{\mathcal{X}}_K} \int_{\hat{K}} \left[\left[\hat{\boldsymbol{u}}^{K,i+1}\right]_{\boldsymbol{n},v} \hat{\phi}_{\boldsymbol{n}} \boldsymbol{e}_v \right]_j \left[\hat{\phi}_{\boldsymbol{\alpha}} \boldsymbol{e}_\gamma\right]_j \, d\boldsymbol{\xi} = \\
&J_{\boldsymbol{\mathcal{X}}_K} \sum_{\boldsymbol{\alpha}' \in \boldsymbol{\mathcal{N}}} \left(\hat{\omega}_{\boldsymbol{\alpha}'} \left[\hat{\boldsymbol{u}}^{K,i+1}\right]_{\boldsymbol{n},v} \hat{\phi}_{\boldsymbol{n}}(\hat{\boldsymbol{\xi}}_{\boldsymbol{\alpha}'}) \left[\boldsymbol{e}_v\right]_j \hat{\phi}_{\boldsymbol{\alpha}}(\hat{\boldsymbol{\xi}}_{\boldsymbol{\alpha}'}) \left[\boldsymbol{e}_\gamma\right]_j \right) = \\
&J_{\boldsymbol{\mathcal{X}}_K} \sum_{\boldsymbol{\alpha}' \in \boldsymbol{\mathcal{N}}} \left(\hat{\omega}_{\boldsymbol{\alpha}'} \left[\hat{\boldsymbol{u}}^{K,i+1}\right]_{\boldsymbol{n},v} \delta_{\boldsymbol{n}\boldsymbol{\alpha}'} \delta_{vj} \delta_{\boldsymbol{\alpha}\boldsymbol{\alpha}'} \delta_{\gamma j} \right) = \\
&J_{\boldsymbol{\mathcal{X}}_K} \hat{\omega}_{\boldsymbol{\alpha}} \left[\hat{\boldsymbol{u}}^{K,i+1}\right]_{\boldsymbol{\alpha},\gamma} .
\end{aligned}
\tag{2.98}
$$

Term U-II

Analogously to the first term of eq. (2.97), the second term can be rewritten as follows:

$$
\begin{aligned}
&\int_K \left[\left[\hat{\boldsymbol{u}}^{K,i}\right]_{\boldsymbol{n},v} \boldsymbol{\phi}^K_{\boldsymbol{n},v} \right]_j \left[\boldsymbol{\phi}^K_{\boldsymbol{\alpha},\gamma}\right]_j \, d\boldsymbol{x} = \\
&J_{\boldsymbol{\mathcal{X}}_K} \hat{\omega}_{\boldsymbol{\alpha}} \left[\hat{\boldsymbol{u}}^{K,i}\right]_{\boldsymbol{\alpha},\gamma} .
\end{aligned}
\tag{2.99}
$$

Term U-III

The third term of eq. (2.97) can be rewritten with respect to reference coordinates as follows:

$$
\begin{aligned}
&\int_{t_i}^{t_i+\Delta t_i}\int_K \left[\boldsymbol{F}\left(\left[\hat{\boldsymbol{q}}^{K,i}\right]_{\boldsymbol{n},l,v}\boldsymbol{\theta}^{Ki}_{\boldsymbol{n},l,v}\right)\right]_{jk}\frac{\partial}{\partial x_k}\left[\boldsymbol{\phi}^K_{\boldsymbol{\alpha},\gamma}\right]_j \, d\boldsymbol{x}dt = \\
&\int_{t_i}^{t_i+\Delta t_i}\int_K \left[\boldsymbol{F}\left(\left[\hat{\boldsymbol{q}}^{K,i}\right]_{\boldsymbol{n},l,v}\phi^K_{\boldsymbol{n}}\psi^i_l\boldsymbol{e}_v\right)\right]_{jk}\frac{\partial}{\partial x_k}\left(\prod_{d=1}^{D}\psi^K_{[\boldsymbol{\alpha}]_d}([\boldsymbol{x}]_d)\right)\left[\boldsymbol{e}_\gamma\right]_j \, d\boldsymbol{x}dt = \\
&\int_{t_i}^{t_i+\Delta t_i}\int_K \left[\boldsymbol{F}\left(\left[\hat{\boldsymbol{q}}^{K,i}\right]_{\boldsymbol{n},v,l}\phi^K_{\boldsymbol{n}}\psi^i_l\boldsymbol{e}_v\right)\right]_{jk}\left(\prod_{d=1,d\neq k}^{D}\psi^K_{[\boldsymbol{\alpha}]_d}([\boldsymbol{x}]_d)\right)\frac{1}{\left[\Delta\boldsymbol{x}^K\right]_k}\frac{\partial}{\partial\xi_k}\hat{\psi}_{[\boldsymbol{\alpha}]_k}\left(\left[\boldsymbol{\mathcal{X}}_K(\boldsymbol{x})\right]_k\right)\left[\boldsymbol{e}_\gamma\right]_j \, d\boldsymbol{x}dt \\
&J_{\mathcal{T}_i}J_{\boldsymbol{\mathcal{X}}_K}\int_0^1\int_{\hat{K}}\left[\boldsymbol{F}\left(\left[\hat{\boldsymbol{q}}^{K,i}\right]_{\boldsymbol{n},l,v}\hat{\phi}_{\boldsymbol{n}}\hat{\psi}_l\boldsymbol{e}_v\right)\right]_{kj}\left(\prod_{d=1,d\neq k}^{D}\hat{\psi}[\boldsymbol{\alpha}]_d(\left[\boldsymbol{\xi}\right]_d)\right)\frac{1}{\left[\Delta\boldsymbol{x}^K\right]_k}\frac{\partial}{\partial\xi_k}\hat{\psi}_{[\boldsymbol{\alpha}]_k}\left(\left[\boldsymbol{\xi}\right]_k\right)\left[\boldsymbol{e}_\gamma\right]_j \, d\boldsymbol{\xi}d\tau = \\
&J_{\mathcal{T}_i}J_{\boldsymbol{\mathcal{X}}_K}\sum_{\boldsymbol{\alpha}'\in\boldsymbol{\mathcal{N}}}\sum_{\beta'\in\mathcal{N}}\left(\hat{\omega}_{\boldsymbol{\alpha}'}\hat{\omega}_{\beta'}\left[\boldsymbol{F}\left(\left[\hat{\boldsymbol{q}}^{K,i}\right]_{\boldsymbol{n},l,v}\hat{\phi}_{\boldsymbol{n}}(\boldsymbol{\xi}_{\boldsymbol{\alpha}'})\hat{\psi}(\hat{\tau}_{\beta'})\boldsymbol{e}_v\right)\right]_{jk}\left(\prod_{d=1,d\neq k}^{D}\hat{\psi}_{[\boldsymbol{\alpha}]_d}(\left[\boldsymbol{\xi}_{\boldsymbol{\alpha}'}\right]_d)\right)\frac{1}{\left[\Delta\boldsymbol{x}^K\right]_k}\frac{\partial}{\partial\xi_k}\hat{\psi}\right. \\
&J_{\mathcal{T}_i}J_{\boldsymbol{\mathcal{X}}_K}\sum_{\boldsymbol{\alpha}'\in\boldsymbol{\mathcal{N}}}\sum_{\beta'\in\mathcal{N}}\left(\hat{\omega}_{\boldsymbol{\alpha}'}\hat{\omega}_{\beta'}\left[\boldsymbol{F}\left(\left[\hat{\boldsymbol{q}}^{K,i}\right]_{\boldsymbol{n},l,v}\delta_{\boldsymbol{n}\boldsymbol{\alpha}'}\delta_{l\beta'}\boldsymbol{e}_v\right)\right]_{jk}\left(\prod_{d=1,d\neq k}^{D}\delta_{[\boldsymbol{\alpha}]_d\left[\boldsymbol{\alpha}'\right]_d}\right)\frac{1}{\left[\Delta\boldsymbol{x}^K\right]_k}\frac{\partial}{\partial\xi_k}\hat{\psi}_{[\boldsymbol{\alpha}]_k}\left(\left[\boldsymbol{\xi}_{\boldsymbol{\alpha}'}\right]_k\right)\right. \\
&J_{\mathcal{T}_i}J_{\boldsymbol{\mathcal{X}}_K}\hat{\omega}_{\boldsymbol{\alpha}}\sum_{k=1}^{D}\left(\sum_{\alpha'_k\in\mathcal{N}}\sum_{\beta'\in\mathcal{N}}\left(\frac{\hat{\omega}_{\beta'}}{\hat{\omega}_{\alpha'_k}}\frac{1}{\left[\Delta\boldsymbol{x}^K\right]_k}\frac{\partial}{\partial\xi_k}\hat{\psi}_{\alpha'_k}\left(\left[\hat{\boldsymbol{\xi}}\right]_{\alpha'_k}\right)\left[\boldsymbol{F}\left(\left[\hat{\boldsymbol{q}}^{K,i}\right]_{\left[[\boldsymbol{\alpha}]_1,[\boldsymbol{\alpha}]_2,\ldots,[\boldsymbol{\alpha}]_{k-1},\alpha'_k,[\boldsymbol{\alpha}]_{k+1},\ldots,[\boldsymbol{\alpha}]_D\right],\beta',v}\boldsymbol{e}\right.\right.\right.\right. \\
&J_{\mathcal{T}_i}J_{\boldsymbol{\mathcal{X}}_K}\hat{\omega}_{\boldsymbol{\alpha}}\sum_{k=1}^{D}\left(\sum_{\alpha'_k\in\mathcal{N}}\sum_{\beta'\in\mathcal{N}}\left(\frac{1}{\hat{\omega}_{\alpha'_k}}\frac{1}{\left[\Delta\boldsymbol{x}^K\right]_k}\left[\boldsymbol{K}\right]_{\alpha'_k,k}\left[\boldsymbol{F}\left(\left[\hat{\boldsymbol{q}}^{K,i}\right]_{\left[[\boldsymbol{\alpha}]_1,[\boldsymbol{\alpha}]_2,\ldots,[\boldsymbol{\alpha}]_{k-1},\alpha'_k,[\boldsymbol{\alpha}]_{k+1},\ldots,[\boldsymbol{\alpha}]_D\right],\beta',v}\boldsymbol{e}_v\right)\right]_{\gamma,k}\right)\right),
\end{aligned}
\tag{2.100}
$$

where we made use of the fact that du to the chain rule:

$$
\begin{aligned}
&\frac{\partial}{\partial x_k}\left(\prod_{d=1}^{D}\psi^K_{[\boldsymbol{\alpha}]_d}([\boldsymbol{x}]_d)\right) = \left(\prod_{d=1,d\neq k}^{D}\psi^K_{[\boldsymbol{\alpha}]_d}([\boldsymbol{x}]_d)\right)\frac{\partial}{\partial x_k}\psi^K_{[\boldsymbol{\alpha}]_k}([\boldsymbol{x}]_k) = \\
&\left(\prod_{d=1,d\neq k}^{D}\psi^K_{[\boldsymbol{\alpha}]_d}([\boldsymbol{x}]_d)\right)\frac{\partial}{\partial \xi_j}\hat{\psi}_{[\boldsymbol{\alpha}]_k}\left(\left[\boldsymbol{\mathcal{X}}_K(\boldsymbol{x})\right]_k\right)\frac{\partial}{\partial x_k}\left[\boldsymbol{\mathcal{X}}_K(\boldsymbol{x})\right]_j = \\
&\left(\prod_{d=1,d\neq k}^{D}\psi^K_{[\boldsymbol{\alpha}]_d}([\boldsymbol{x}]_d)\right)\frac{\partial}{\partial \xi_j}\hat{\psi}_{[\boldsymbol{\alpha}]_k}\left(\left[\boldsymbol{\mathcal{X}}_K(\boldsymbol{x})\right]_k\right)\frac{1}{\left[\Delta\boldsymbol{x}^K\right]_k}\delta_k j = \\
&\left(\prod_{d=1,d\neq k}^{D}\psi^K_{[\boldsymbol{\alpha}]_d}([\boldsymbol{x}]_d)\right)\frac{1}{\left[\Delta\boldsymbol{x}^K\right]_k}\frac{\partial}{\partial \xi_k}\hat{\psi}_{[\boldsymbol{\alpha}]_k}\left(\left[\boldsymbol{\mathcal{X}}_K(\boldsymbol{x})\right]_k\right)\, d\boldsymbol{x}dt. \qquad (2.101)
\end{aligned}
$$

Term U-IV

The fourth term of eq. (2.97) can be rewritten with respect to reference coordinates as follows:

$$
\begin{aligned}
&\int_{t_i}^{t_i+\Delta t_i}\int_K\left[\boldsymbol{s}\left(\left[\hat{\boldsymbol{q}}^{K,i}\right]_{\boldsymbol{n},l,v}\boldsymbol{\theta}^{Ki}_{\boldsymbol{n},l,v}\right)\right]_j\left[\boldsymbol{\phi}^K_{\boldsymbol{\alpha},\gamma}\right]_j d\boldsymbol{x}dt = \\
&\int_{t_i}^{t_i+\Delta t_i}\int_K\left[\boldsymbol{s}\left(\left[\hat{\boldsymbol{q}}^{K,i}\right]_{\boldsymbol{n},l,v}\phi^K_{\boldsymbol{n}}\psi^i_l\boldsymbol{e}_v\right)\right]_j\phi^K_{\boldsymbol{\alpha}}\left[\boldsymbol{e}_\gamma\right]_j d\boldsymbol{x}dt = \\
&J_{\mathcal{T}_i}J_{\boldsymbol{\mathcal{X}}_K}\int_0^1\int_{\hat{K}}\left[\boldsymbol{s}\left(\left[\hat{\boldsymbol{q}}^{K,i}\right]_{\boldsymbol{n},l,v}\hat{\phi}_{\boldsymbol{n}}\hat{\psi}_l\boldsymbol{e}_v\right)\right]_j\hat{\phi}_{\boldsymbol{\alpha}}\left[\boldsymbol{e}_\gamma\right]_j d\boldsymbol{\xi}d\tau = \\
&J_{\mathcal{T}_i}J_{\boldsymbol{\mathcal{X}}_K}\sum_{\boldsymbol{\alpha}'\in\boldsymbol{\mathcal{N}}}\sum_{\beta'\in\mathcal{N}}\left(\hat{\omega}_{\boldsymbol{\alpha}'}\hat{\omega}_{\beta'}\left[\boldsymbol{s}\left(\left[\hat{\boldsymbol{q}}^{K,i}\right]_{\boldsymbol{n},l,v}\hat{\phi}_{\boldsymbol{n}}(\hat{\boldsymbol{\xi}}_{\boldsymbol{\alpha}'})\hat{\psi}_l(\hat{\tau}_{\beta'}\boldsymbol{e}_v)\right)\right]_j\hat{\phi}_{\boldsymbol{\alpha}}(\hat{\boldsymbol{\xi}}_{\boldsymbol{\alpha}'})\left[\boldsymbol{e}_\gamma\right]_j\right) = \\
&J_{\mathcal{T}_i}J_{\boldsymbol{\mathcal{X}}_K}\sum_{\boldsymbol{\alpha}'\in\boldsymbol{\mathcal{N}}}\sum_{\beta'\in\mathcal{N}}\left(\hat{\omega}_{\boldsymbol{\alpha}'}\hat{\omega}_{\beta'}\left[\boldsymbol{s}\left(\left[\hat{\boldsymbol{q}}^{K,i}\right]_{\boldsymbol{n},l,v}\delta_{\boldsymbol{n}\boldsymbol{\alpha}'}\delta_{l\beta'}\boldsymbol{e}_v)\right)\right]_j\delta_{\boldsymbol{\alpha}\boldsymbol{\alpha}'}\delta_{\gamma j}\right) = \\
&J_{\mathcal{T}_i}J_{\boldsymbol{\mathcal{X}}_K}\hat{\omega}_{\boldsymbol{\alpha}}\sum_{\beta'\in\mathcal{N}}\left(\hat{\omega}_{\beta'}\left[\boldsymbol{s}\left(\left[\hat{\boldsymbol{q}}^{K,i}\right]_{\boldsymbol{\alpha},\beta',v}\boldsymbol{e}_v\right)\right]_\gamma\right).
\end{aligned}
\qquad (2.102)
$$

Term U-V

Let $d \in \mathcal{D}$ and $e \in \{0,1\} := \mathcal{E}$. Then if we define the $D-1$-dimensional quadrilateral $\partial\hat{K}_{d,e}$ as

$$\partial\hat{K}_{d,e} = \left\{\boldsymbol{\xi}\in\hat{K}\mid\left[\boldsymbol{\xi}\right]_d = e\right\}, \qquad (2.103)$$

the set $\{\partial\hat{K}_{d,e}\}_{d\in\mathcal{D},e\in\mathcal{E}}$ is a partition of the surface $\partial\hat{K}$ of the spatial reference element. By making use of the mappings $\boldsymbol{\mathcal{X}}_K$ that maps points $\boldsymbol{\xi} \in \hat{K}$ to $\boldsymbol{x} \in K$ for all $K \in \mathcal{K}_h$ we can define

$$\partial K_{d,e} = \boldsymbol{\mathcal{X}}_K\left(\partial\hat{K}_{d,e}\right), \tag{2.104}$$

where now the set $\{\partial K_{d,e}\}_{d\in\mathcal{D},e\in\mathcal{E}}$ is a quadrilateral partition of the surface ∂K for all cells $K \in \mathcal{K}_h$.

In consequence the surface integral in the fifth term of eq. (2.97) can be rewritten as follows:

$$\begin{aligned}
&\int_{t_i}^{t_i+\Delta t_i}\int_{\partial K}\left[\boldsymbol{\mathcal{G}}\left(\hat{\boldsymbol{q}}^{K,i},\hat{\boldsymbol{q}}^{K^+,i},\boldsymbol{n}\right)\right]_j\left[\boldsymbol{\phi}^K_{\boldsymbol{\alpha},\gamma}\right]_j ds(\boldsymbol{x})dt = \\
&\int_{t_i}^{t_i+\Delta t_i}\sum_{d\in\mathcal{D}}\sum_{e\in\mathcal{E}}\left(\int_{\partial K_{d,e}}\left[\boldsymbol{\mathcal{G}}\left(\hat{\boldsymbol{q}}^{K,i},\hat{\boldsymbol{q}}^{K^+,i},\boldsymbol{e}_d\right)\right]_j\phi^K_{\boldsymbol{\alpha}}\left[\boldsymbol{e}_\gamma\right]_j ds(\boldsymbol{x})\right) dt = \\
&J_{\mathcal{T}_i}J_{\boldsymbol{\mathcal{X}}_K}\int_0^1\sum_{d\in\mathcal{D}}\sum_{e\in\mathcal{E}}\left(\frac{1}{\left[\Delta\boldsymbol{x}^K\right]_d}\int_{\partial\hat{K}_{d,e}}\left[\boldsymbol{\mathcal{G}}\left(\hat{\boldsymbol{q}}^{K,i},\hat{\boldsymbol{q}}^{K^+,i},(-1)^e\boldsymbol{e}_d\right)\right]_j\hat{\phi}_{\boldsymbol{\alpha}}\left[\boldsymbol{e}_d\right]_j ds(\boldsymbol{\xi})\right) d\tau = \\
&J_{\mathcal{T}_i}J_{\boldsymbol{\mathcal{X}}_K}\sum_{\beta'\in\mathcal{D}}\hat{\omega}_{\beta'}\sum_{d\in\mathcal{D}}\sum_{e\in\mathcal{E}}\sum_{\boldsymbol{\alpha}'\in\boldsymbol{\mathcal{N}}^-}\left(\hat{\omega}_{\boldsymbol{\alpha}'}\frac{1}{\left[\Delta\boldsymbol{x}^K\right]_d}\left[\boldsymbol{\mathcal{G}}\left(\hat{\boldsymbol{q}}^{K,i},\hat{\boldsymbol{q}}^{K^+,i},(-1)^e\boldsymbol{e}_d\right)\right]_j\hat{\phi}_{\boldsymbol{\alpha}^d}(\hat{\boldsymbol{\xi}}_{\boldsymbol{\alpha}'})\left(\hat{\psi}_{[\boldsymbol{\alpha}]_d}\Big|_e\right)\left[\boldsymbol{e}_d\right]_j\right) = \\
&J_{\mathcal{T}_i}J_{\boldsymbol{\mathcal{X}}_K}\sum_{\beta'\in\mathcal{D}}\hat{\omega}_{\beta'}\sum_{d\in\mathcal{D}}\sum_{e\in\mathcal{E}}\sum_{\boldsymbol{\alpha}'\in\boldsymbol{\mathcal{N}}^-}\left(\hat{\omega}_{\boldsymbol{\alpha}'}\frac{1}{\left[\Delta\boldsymbol{x}^K\right]_d}\left[\boldsymbol{\mathcal{G}}\left(\hat{\boldsymbol{q}}^{K,i},\hat{\boldsymbol{q}}^{K^+,i},(-1)^e\boldsymbol{e}_d\right)\right]_j\delta_{\boldsymbol{\alpha}^d\boldsymbol{\alpha}'}\left(\hat{\psi}_{[\boldsymbol{\alpha}]_d}\Big|_e\right)\delta_{\gamma j}\right) = \\
&J_{\mathcal{T}_i}J_{\boldsymbol{\mathcal{X}}_K}\hat{\omega}_{\boldsymbol{\alpha}}\sum_{\beta'\in\mathcal{D}}\sum_{d\in\mathcal{D}}\sum_{e\in\mathcal{E}}\sum_{\alpha'_d\in\mathcal{N}}\left(\frac{\hat{\omega}_{\beta'}}{\hat{\omega}_{\alpha'_d}}\frac{1}{\left[\Delta\boldsymbol{x}^K\right]_d}\left[\boldsymbol{\mathcal{G}}\left(\hat{\boldsymbol{q}}^{K,i},\hat{\boldsymbol{q}}^{K^+,i},(-1)^e\boldsymbol{e}_d\right)\right]_\gamma\left(\hat{\psi}_{\alpha'_d}\Big|_e\right)\right) = \\
&J_{\mathcal{T}_i}J_{\boldsymbol{\mathcal{X}}_K}\hat{\omega}_{\boldsymbol{\alpha}}\sum_{\beta'\in\mathcal{D}}\sum_{d\in\mathcal{D}}\sum_{e\in\mathcal{E}}\sum_{\alpha'_d\in\mathcal{N}}\left(\frac{\hat{\omega}_{\beta'}}{\hat{\omega}_{\alpha'_d}}\frac{1}{\left[\Delta\boldsymbol{x}^K\right]_d}\left[\boldsymbol{\mathcal{G}}\left(\hat{\boldsymbol{q}}^{K,i},\hat{\boldsymbol{q}}^{K^+,i},(-1)^e\boldsymbol{e}_d\right)\right]_\gamma\left(\delta_{e0}\boldsymbol{l}_{\alpha'_d}\right)\left(\delta_{e1}\boldsymbol{r}_{\alpha'_d}\right)\left(\hat{\psi}_{\alpha'_d}\Big|_e\right)\right).
\end{aligned} \tag{2.105}$$

In each term we have to solve a Riemann problem in direction of the unit vector $\boldsymbol{e}_d$ defined as

$$\left[\boldsymbol{e}_d\right]_{d'} = \delta_{dd'} \tag{2.106}$$

for $d' \in \mathcal{D}$. $\boldsymbol{l}$ and $\boldsymbol{r}$ denote the Left and Right Reference Element Flux Operator, respectively, and the latter is defined as

$$\left[\boldsymbol{r}\right]_i = \hat{\psi}_i\Big|_1 \tag{2.107}$$

for $i \in \mathcal{N}$. A Python script to compute $\boldsymbol{r}$ can be found in appendix A.

The Complete One-step Update Scheme

Inserting eqs. (2.98) to (2.100), (2.102) and (2.105) into eq. (2.97) and dividing the resulting equation by $\hat{\omega}_{\boldsymbol{\alpha}}$ and $J_{\boldsymbol{\mathcal{X}}_K}$ yields

$$\begin{aligned}
&\left[\hat{\boldsymbol{u}}^{K,i+1}\right]_{\boldsymbol{\alpha},\gamma} = \left[\hat{\boldsymbol{u}}^{K,i}\right]_{\boldsymbol{\alpha},\gamma} + \\
&J_{\mathcal{T}_i} \sum_{k=1}^{D} \left(\sum_{\alpha'_k \in \mathcal{N}} \sum_{\beta' \in \mathcal{N}} \left(\frac{\hat{\omega}_{\beta'}}{\hat{\omega}_{\alpha'_k}} \frac{1}{\left[\Delta \boldsymbol{x}^K\right]_k} \underbrace{\frac{\partial}{\partial \xi_k} \hat{\psi}_{\alpha'_k} \left(\left[\hat{\boldsymbol{\xi}}\right]_{\alpha'_k} \right)}_{\mathrm{Kxi}_{\alpha'_k k}} \left[\boldsymbol{F} \left(\left[\hat{\boldsymbol{q}}^{K,i}\right]_{\left[[\boldsymbol{\alpha}]_1,[\boldsymbol{\alpha}]_2,\ldots,[\boldsymbol{\alpha}]_{k-1},\alpha'_k,[\boldsymbol{\alpha}]_{k+1},\ldots,[\boldsymbol{\alpha}]_D\right],\beta',v} \boldsymbol{e}_v \right) \right]_{\gamma,k} \right) \right) + \\
&J_{\mathcal{T}_i} \sum_{\beta' \in \mathcal{N}} \left(\hat{\omega}_{\beta'} \left[\boldsymbol{s} \left(\left[\hat{\boldsymbol{q}}^{K,i}\right]_{\boldsymbol{\alpha},\beta',v} \boldsymbol{e}_v \right) \right]_{\gamma} \right) - \\
&J_{\mathcal{T}_i} \sum_{\beta' \in \mathcal{D}} \sum_{d \in \mathcal{D}} \sum_{e \in \mathcal{E}} \sum_{\alpha'_d \in \mathcal{N}} \left(\frac{\hat{\omega}_{\beta'}}{\hat{\omega}_{\alpha'_d}} \frac{1}{\left[\Delta \boldsymbol{x}^K\right]_d} \left[\boldsymbol{\mathcal{G}} \left(\hat{\boldsymbol{q}}^{K,i}, \hat{\boldsymbol{q}}^{K^+,i}, (-1)^e \boldsymbol{e}_d \right) \right]_{\gamma} \underbrace{\left(\hat{\psi}_{\alpha'_d} \Big|_e \right)}_{\mathrm{F0,\ F1}} \right),
\end{aligned} \tag{2.108}$$

which we require to hold for $\boldsymbol{\alpha} \in \boldsymbol{\mathcal{N}}$, $\gamma \in \mathcal{V}$, $K \in \mathcal{K}_h$ and $i \in \mathcal{I}$.

Time step restriction

For the scheme to be stable we require that the following inequality holds for the time step Δt_i:

$$\Delta t_i \leq \frac{1}{D} \frac{1}{(2N+1)} \min_{d \in \mathcal{D}} \left(\frac{[\Delta \boldsymbol{x}]_d}{\Lambda^{i,d}} \right), \tag{2.109}$$

where

$$\Lambda^{i,d} = \operatorname*{maxabs}_{v \in \mathcal{V}} \left[\boldsymbol{\lambda}^{i,d}\right]_v \tag{2.110}$$

and $\boldsymbol{\lambda}^{d,i}$ is a vector containing the V real eigenvalues of the Jacobian

$$\frac{\partial}{\partial x_k} \left[\boldsymbol{F} \left(\boldsymbol{u}(\boldsymbol{x}, t_i) \right) \right]_{jd} \tag{2.111}$$

for the respective dimension $d \in \mathcal{D}$ and the index of the current time step $i \in \mathcal{I}$. For details on the derivation of this formula see [10, 3, 11].

2.1.13 A Posteriori Subcell Limiting

The unlimited ADER-DG scheme derived in the previous section allows us to solve non-linear hyperbolic balance laws with arbitrary order in both time and space for continuous data. For discontinuous initial data and in scenarios where even for smooth initial data due to the non-linear nature of the system shocks arise, however, our high-order discontinuous Galerkin method is unsuitable. It is a linear scheme in the sense of Godunov, i.e. the coefficient of the scheme are independent of the current state of the system (see [12, 2] for a derivation of the so-called Godunov theorem) and therefore shocks in the system do not only give rise to artificial and persistent oscillations, but also decrease pointwise accuracy in the vicinity to first order and even cause loss of pointwise convergence at the point of discontinuity (see [6] for details). The quality of the numerical solution is unacceptable at best, if not the simulation crashes altogether due to a strictly positive physical quantity such as pressure or density becoming negative and thereby invalidating the well-posedness of the problem.

A way to solving these issue is limiting. For an exhaustive overview on classical limiting approaches see again [2]. According to Dumbser et al. ([3]), there are two key challenges in designing limiter procedures:

1. A so-called troubled cell indicator needs to implement criteria on how to identify troubled cells, i.e. cells that need limiting.

2. The troubled DG solution needs to be replaced by a robust non-linear reconstructions scheme in a way such that the minimum amount of artificial numerical viscosity is injected in these cells and such that the subcell resolution property of the DG scheme is conserved.

In the following we will present a novel approach called a-posterior subcell limiting. It was first introduced in 2014 (see [3]) and has very favorable properties with respect to the aforementioned challenges. The general idea is to first project the local DG solution onto a finer equidistant grid to check if the solution satisfies certain admissibility criteria. If this is not the case, we discard the DG candidate solution and instead rely on the more robust MUSCL-Hancock FVM scheme starting from the fine grid solution of the previous time step (which is either the projected DG solution or, if the cell has already been troubled in the previous time step, the FVM fine grid solution). For troubled cells we finally use a reconstruction operator to replace the rejected candidate solution by the transformed fine grid solution. The remainder of this chapter is structured as follows: We will first introduce the necessary projection and reconstruction operators. We will then discuss a possible set of admissibility criteria and conclude with a summary on the MUSCL-Hancock FVM scheme.

Projection and Reconstruction

In order to check for admissibility and in case of a troubled cell to employ a more robust FVM scheme, we need to project the ADER-DG degrees of freedom $\hat{\boldsymbol{u}}^{K,i}$ to N_S^D subcell averages $\hat{\boldsymbol{p}}^{K,i}$ on an equidistant fine grid. We choose to split $K \in \mathcal{K}_h$ into $N_S = 2N + 1$ subcells along each spatial dimension, creating a total of N_S^D subcells denoted as $K_{\boldsymbol{\alpha}}$, $\boldsymbol{\alpha} \in \{0, 1, \ldots, 2N\} := \boldsymbol{\mathcal{N}_S}$. For explicit Godunov-type finite volume schemes on the subgrid we must satisfy the stability condition

$$\Delta t \leq \frac{1}{d}\frac{1}{N_S}\min_{d\in\mathcal{D}}\left(\frac{[\Delta \boldsymbol{x}]_d}{\Lambda^d}\right). \tag{2.112}$$

Comparing eq. (2.112) to the time step restriction for the ADER-DG scheme given in eq. (2.109) illustrates that the choice $N_S = 2N + 1$ is optimal in the sense that it makes sure that, on the one hand, time steps on the ADER-DG grid are also stable on the equidistant subgrid and, on the other hand, that we add the minimum amount of dissipation necessary. See [3] for additional remarks on the optimality of this particular choice of N_S.

Let as before $K_{\boldsymbol{\alpha}}$ be a cell in the equidistant subgrid on a cell $K \in \mathcal{K}_h$ with cardinality $N_S^D = (2N + 1)^D$. Then we can define a projection $\boldsymbol{\mathcal{P}}$ of the degrees of freedom from the ADER-DG grid cell K to the subcell $K_{\boldsymbol{\alpha}}$ for all $\boldsymbol{\alpha} \in \boldsymbol{\mathcal{N}_S}$ by demanding that the integral averages over $K_{\boldsymbol{\alpha}}$ are preserved. Mathematically this requirement can be expressed as

$$\begin{aligned}
\left[\hat{\boldsymbol{p}}^{K,i}\right]_{\boldsymbol{\alpha},\gamma} =& \frac{1}{|K_{\boldsymbol{\alpha}}|}\int_{K_{\boldsymbol{\alpha}}}\left[\tilde{\boldsymbol{u}}_h^{K,i}\right]_\gamma d\boldsymbol{x} = \frac{1}{|K_{\boldsymbol{\alpha}}|}\int_{K_{\boldsymbol{\alpha}}}\left[\hat{\boldsymbol{u}}^{K,i}\right]_{\boldsymbol{n},v}\phi_{\boldsymbol{n}}^K(\boldsymbol{x})\left[\boldsymbol{e}_v\right]_\gamma d\boldsymbol{x} = \\
& \sum_{\boldsymbol{n}\in\boldsymbol{\mathcal{N}}}\sum_{\boldsymbol{\alpha}'\in\boldsymbol{\mathcal{N}}}\left(\left[\hat{\boldsymbol{u}}^{K,i}\right]_{\boldsymbol{n},\gamma}\hat{\omega}_{\boldsymbol{\alpha}'}\hat{\phi}_{\boldsymbol{n}}\left(\frac{1}{N_S}\boldsymbol{\alpha}+\frac{1}{N_S}\hat{\boldsymbol{\xi}}_{\boldsymbol{\alpha}'}\right)\right) = \\
& \sum_{\boldsymbol{n}\in\boldsymbol{\mathcal{N}}}\sum_{\boldsymbol{\alpha}'\in\boldsymbol{\mathcal{N}}}\left(\prod_{d\in\mathcal{D}}\left(\hat{\omega}_{\boldsymbol{\alpha}'_d}\hat{\psi}_{\boldsymbol{n}_d}\left(\frac{1}{N_S}\boldsymbol{\alpha}_d+\frac{1}{N_S}\hat{\xi}_{\boldsymbol{\alpha}'_d}\right)\right)\left[\hat{\boldsymbol{u}}^{K,i}\right]_{\boldsymbol{n},\gamma}\right) = \\
& \sum_{\boldsymbol{n}\in\boldsymbol{\mathcal{N}}}\left(\sum_{\alpha'_0\in\mathcal{N}}\left(\hat{\omega}_{\alpha'_0}\hat{\psi}_{\boldsymbol{n}_0}\left(\frac{1}{N_S}\boldsymbol{\alpha}_0+\frac{1}{N_S}\hat{\xi}_{\alpha'_0}\right)\ldots\sum_{\alpha'_{D-1}}\left(\hat{\omega}_{\alpha'_{D-1}}\hat{\psi}_{\boldsymbol{n}_{D-1}}\left(\frac{1}{N_S}\boldsymbol{\alpha}_{D-1}+\frac{1}{N_S}\hat{\xi}_{\alpha'_{D-1}}\right)\right)\right)\right)\left[\hat{\boldsymbol{u}}^{K,i}\right]_{\boldsymbol{n},} \\
& \sum_{\boldsymbol{n}\in\boldsymbol{\mathcal{N}}}\left(\prod_{d\in\mathcal{D}}\left(\left[\boldsymbol{P}\right]_{\boldsymbol{\alpha}_d,\boldsymbol{n}_d}\right)\left[\hat{\boldsymbol{u}}^{K,i}\right]_{\boldsymbol{n},\gamma}\right),
\end{aligned} \tag{2.113}$$

where $|K_{\boldsymbol{\alpha}}|$ denotes the volume of $K_{\boldsymbol{\alpha}}$ for $\boldsymbol{\alpha} \in \boldsymbol{\mathcal{N}}$ and $\gamma \in \mathcal{D}$. The computation can be carried out efficiently in a dimension-by-dimension manner using the projection matrix

$$[\boldsymbol{P}]_{ij} = \sum_{k \in \mathcal{N}} \left(\hat{\omega}_k \hat{\psi}_j \left(\frac{1}{N_S} i + \frac{1}{N_S} \hat{\xi}_k \right) \right) \tag{2.114}$$

for $i \in \mathcal{N}_{\mathcal{S}}$ and $j \in \mathcal{N}$. A Python script to compute the projection matrix $\boldsymbol{P}$ can be found in appendix A.

To enable replacement of the invalid candidate solution $\hat{\boldsymbol{u}}^{K,i}$ we need to define a reconstruction operator $\boldsymbol{\mathcal{R}}$ which transforms the degrees of freedom $\hat{\boldsymbol{p}}^{K,i}$ of the solution computed by the FVM scheme back to the ADER-DG grid. Since we have more degrees of freedom on the fine grid than on the ADER-DG grid, we can now only require that the constrains based on preservation of local averages, i.e.

$$\frac{1}{|K_{\boldsymbol{\alpha}}|} \int_{K_{\boldsymbol{\alpha}}} \left[\tilde{\boldsymbol{u}}_h^{K,i} \right]_\gamma \, d\boldsymbol{x} = \frac{1}{|K_{\boldsymbol{\alpha}}|} \int_{K_{\boldsymbol{\alpha}}} \left[\tilde{\boldsymbol{p}}^{K,i} \right]_\gamma \, d\boldsymbol{x} = \left[\hat{\boldsymbol{p}}^{K,i} \right]_{\boldsymbol{\alpha},\gamma} \tag{2.115}$$

for $\boldsymbol{\alpha} \in \mathcal{N}_{\mathcal{S}}$ and $\gamma \in \mathcal{V}$, are fulfilled in a least squares sense. We do however insist that the overall integral average

$$\int_K \left[\tilde{\boldsymbol{p}}^{K,i} \right]_\gamma \, d\boldsymbol{x} = \int_K \left[\tilde{\boldsymbol{u}}_h^{K,i} \right]_\gamma \, d\boldsymbol{x}. \tag{2.116}$$

are preserved to make sure that the scheme remains conservative.

Analogously to eq. **??** the left-hand side of eq. (2.115) can be simplified as follows:

$$\begin{aligned} &\frac{1}{K_{\boldsymbol{\alpha}}} \int_{K_{\boldsymbol{\alpha}}} \left[\hat{\boldsymbol{u}}^{K,i} \right]_{\boldsymbol{n},\gamma} \phi_{\boldsymbol{n}}^K \, d\boldsymbol{x} = \\ &\int_{\hat{K}} \left[\hat{\boldsymbol{u}}^{K,i} \right]_{\boldsymbol{n},\gamma} \hat{\phi}_{\boldsymbol{n}} \left(\frac{1}{N_S} \boldsymbol{\alpha} + \frac{1}{N_S} \boldsymbol{\xi} \right) \, d\boldsymbol{\xi} = \\ &\sum_{\boldsymbol{n} \in \boldsymbol{\mathcal{N}}} \left(\prod_{d \in \mathcal{D}} \left([\boldsymbol{P}]_{\alpha_d, n_d} \right) \left[\hat{\boldsymbol{u}}^{K,i} \right]_{\boldsymbol{n},\gamma} \right). \end{aligned} \tag{2.117}$$

Eq. (2.116) simplifies to

$$\sum_{\boldsymbol{\alpha} \in \mathcal{N}_{\mathcal{S}}} \frac{1}{|K_{\boldsymbol{\alpha}}|} \left[\hat{\boldsymbol{p}}^{K,i} \right]_{\boldsymbol{\alpha},v} = \frac{1}{N_S^D} \sum_{\boldsymbol{\alpha} \in \mathcal{N}_{\mathcal{S}}} \left[\hat{\boldsymbol{p}}^{K,i} \right]_{\boldsymbol{\alpha},v} = \sum_{\boldsymbol{n} \in \boldsymbol{\mathcal{N}}} \hat{\omega}_{\boldsymbol{n}} \left[\hat{\boldsymbol{u}}^{K,i} \right]_{\boldsymbol{n},v} \tag{2.118}$$

for $v \in V$. In order to be able to carry out the computation in a dimension-by-dimension manner as for the projection we need to solve the following constrained least squares optimization problem to find the reconstruction matrix $\boldsymbol{R} \in \mathbb{R}^{(N+1)\times(N+1)}$ for $\boldsymbol{P} \in \mathbb{R}^{N_S\times(N+1)}$ being the projection matrix and for vectors $\boldsymbol{p} \in \mathbb{R}^{N_S}$ and $\boldsymbol{u} \in \mathbb{R}^{N+1}$ representing the degrees of freedom along a coordinate axis for a single quantity:

$$\begin{aligned} &\underset{\boldsymbol{u}\in\mathbb{R}^{N+1}}{\text{minimize}} && \| [\boldsymbol{P}]_{ij} [\boldsymbol{u}]_j - [\boldsymbol{p}]_j \|^2 \\ &\text{subject to} && \hat{\omega}_i [\boldsymbol{u}]_i = \frac{1}{N_S} \sum_i [\boldsymbol{p}]_i \,. \end{aligned} \tag{2.119}$$

The corresponding Lagrange dual problem (see e.g. [13] for a general derivation) written in matrix vector notation reads

$$\begin{bmatrix} 2\boldsymbol{P}^T\boldsymbol{P} & \hat{\boldsymbol{\omega}}^T \\ \hat{\boldsymbol{\omega}} & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{u} \\ z \end{bmatrix} = \begin{bmatrix} 2\boldsymbol{P}^T \\ \frac{1}{N_S}\boldsymbol{1}^T \end{bmatrix} \begin{bmatrix} \boldsymbol{p} \end{bmatrix}, \tag{2.120}$$

where $z \in \mathbb{R}$ is a scalar Lagrange multiplier and $\boldsymbol{1}$ is a vector with all components equal to one. We can then compute

$$\begin{bmatrix} \tilde{\boldsymbol{R}} \end{bmatrix} = \begin{bmatrix} 2\boldsymbol{P}^T\boldsymbol{P} & \hat{\boldsymbol{\omega}}^T \\ \hat{\boldsymbol{\omega}} & 0 \end{bmatrix}^{-1} \begin{bmatrix} 2\boldsymbol{P}^T \\ \frac{1}{N_S}\boldsymbol{1}^T \end{bmatrix}. \tag{2.121}$$

Since we are only interested in the first $N+1$ rows of $\tilde{\boldsymbol{R}}$ the reconstruction matrix $\boldsymbol{R}$ is obtained after discarding the last row, i.e.

$$\begin{bmatrix} \boldsymbol{R} \end{bmatrix}_{0:N,0:N_S-1} = \begin{bmatrix} \tilde{\boldsymbol{R}} \end{bmatrix}_{0:N,0:N_S-1}. \tag{2.122}$$

A Python notebook to compute $\boldsymbol{R}$ can be found in appendix A. For more details on the transformations see [14].

Identification of Troubled Cells

Once a candidate solution now denoted as $\tilde{\boldsymbol{u}}_h^{K,i+1^*}$ represented by coefficients $\hat{\boldsymbol{u}}^{K,i+1^*}$ in a cell $K \in \mathcal{K}_h$ provided by the unlimited ADER-DG scheme is available, we can apply the set of rules defined by the troubled cell indicator to check if limiting is required. Following the terminology introduced in [3] we discriminate between two kinds of detection criteria:

1. Physical admissibility detection (PAD) can be used to incorporate domain knowledge into the simulation. Most commonly criteria of this kind are used to enforce that quantities with physical meaning remain within their respective domain of definition so that the well-posedness of the problem remains ensured. A PAD criterion is represented by a set of J inequalities of the following form:

$$\pi_j(\tilde{\boldsymbol{u}}_h^{K,i+1^*}) > 0. \tag{2.123}$$

A candidate solution in a cell $K \in \mathcal{K}_h$ is considered admissible in a physical sense if all J inequalities are fulfilled. Considering the Euler equations as an illustrative example, if we want to ensure positivity of the pressure p and the density ρ, we will have two inequalities with $\pi_1(\tilde{\boldsymbol{u}}_h^{K,i+1^*}) = p$ and $\pi_2(\tilde{\boldsymbol{u}}_h^{K,i+1^*}) = \rho$.

2. Numerical admissibility detection (NAD) in the form of a relaxed version of the discrete minimum principle (DMP) ensures that the total variation of the solution remains bounded. In theory we would like to require that

$$\min_{\boldsymbol{y} \in V(K)} \left[\tilde{\boldsymbol{u}}_h^{K',i}(\boldsymbol{y}, t_i)\right]_v - \boldsymbol{\delta}_v \leq \left[\tilde{\boldsymbol{u}}_h^{K,i+1^*}(\boldsymbol{x}, t_{i+1})\right]_v \leq \max_{\boldsymbol{y} \in V(K)} \left[\tilde{\boldsymbol{u}}_h^{K',i}(\boldsymbol{y}, t_i)\right]_v + \boldsymbol{\delta}_v \tag{2.124}$$

holds for all $v \in \mathcal{V}$, $\boldsymbol{x} \in K$, $K \in \mathcal{K}_h$, i.e. the values of all quantities inside K in the candidate solution should be bounded by the solution on the Voronoi neighbors in the previous time step. The vector $\boldsymbol{\delta}$ defined as

$$\boldsymbol{\delta}_v = \varepsilon \left(\max_{\boldsymbol{y} \in V(K)} \left[\tilde{\boldsymbol{u}}_h^{K',i}(\boldsymbol{y}, t_i)\right]_v - \min_{\boldsymbol{y} \in V(K)} \left[\tilde{\boldsymbol{u}}_h^{K',i}(\boldsymbol{y}, t_i)\right]_v \right) \tag{2.125}$$

for example with $\varepsilon = 10^3$ to avoid problems with floating point errors. Since it would be prohibitively expensive to compute the extrema for all polynomials, we settle for

$$\min_{\substack{K' \in V(K),\\ \boldsymbol{\alpha}' \in \boldsymbol{\mathcal{N}_S}}} \left[\hat{\boldsymbol{p}}^{K',i}\right]_{\boldsymbol{\alpha}',v} - \boldsymbol{\delta}_v \leq \left[\hat{\boldsymbol{p}}^{K,i+1^*}\right]_{\boldsymbol{\alpha},v} \leq \max_{\substack{K' \in V(K),\\ \boldsymbol{\alpha}' \in \boldsymbol{\mathcal{N}_S}}} \left[\hat{\boldsymbol{p}}^{K',i}\right]_{\boldsymbol{\alpha}',v} - \boldsymbol{\delta}_v \tag{2.126}$$

instead, i.e. we enforce the relaxed DMP criterion only for the subcell averages on the refined equidistant grid. We apply the projection operator to obtain $\hat{\boldsymbol{p}}^{K,i+1^*} = \boldsymbol{\mathcal{P}}(\hat{\boldsymbol{u}}^{K,i+1^*})$.

The MUSCL-Hancock Scheme

For the sake of completeness we will now give a quick summary on the nonlinear (in the sense of Godunov), second order accurate MUSCL[3]-Hancock FVM scheme. Of course this is only one possible choice for a robust, lower-order scheme that can be used as a "fallback" in the a posterior subcell limiting approach. Following the recipe given in [2] and modified to incorporate source terms and an arbitrary number of space dimensions, the scheme consists of the following steps:

[3] Monotonic Upstream-Centered Scheme for Conservation Laws

1. **Compute slopes:**

$$\left[\hat{\boldsymbol{\delta}}^{K,i}\right]_{d,\boldsymbol{\alpha},\gamma} = \text{minmod}\left(\left[\hat{\boldsymbol{p}}^{K,i}\right]_{\boldsymbol{\alpha}+\boldsymbol{e}_d,\gamma} - \left[\hat{\boldsymbol{p}}^{K,i}\right]_{\boldsymbol{\alpha},\gamma}, \left[\hat{\boldsymbol{p}}^{K,i}\right]_{\boldsymbol{\alpha},\gamma} - \left[\hat{\boldsymbol{p}}^{K,i}\right]_{\boldsymbol{\alpha}-\boldsymbol{e}_d,\gamma}\right) \tag{2.127}$$

for all subgrid cells $K_{\boldsymbol{\alpha}} \subset K$, $\boldsymbol{\alpha} \in \{0,1,\dots,N_S+1\}^D := \boldsymbol{\mathcal{N}}^*_{\boldsymbol{\mathcal{S}}}$, $\gamma \in \mathcal{V}$, troubled grid cells $K \in \mathcal{K}^*_h$, unit vectors $\boldsymbol{e}_d$ and $d \in \mathcal{D}$. A component of the index $\boldsymbol{\alpha} \in \boldsymbol{\mathcal{N}}^*_{\boldsymbol{\mathcal{S}}}$ being 0 or N_S+1 implies access into $\hat{\boldsymbol{p}}^{K',i}$, i.e. access into the fine grid solution on the respective Voroni neighbor $K' \in V(K)$ of K. We furthermore use the common definition of the minmod function, namely

$$\text{minmod}(a,b) = \begin{cases} 0 & \text{if } ab \leq 0 \\ a & \text{if } ab > 0 \text{ and } |a| \leq |b| \\ b & \text{if } ab > 0 \text{ and } |b| < |a|. \end{cases} \tag{2.128}$$

2. **Evaluate source:**

$$\left[\hat{\boldsymbol{s}}^{K,i}\right]_{\boldsymbol{\alpha},\gamma} = \left[\boldsymbol{s}\left(\left[\hat{\boldsymbol{q}}^{K,i}\right]_{\alpha}\right)\right]_{\gamma} \tag{2.129}$$

for all subgrid cells $K_{\boldsymbol{\alpha}} \subset K$, $\boldsymbol{\alpha} \in \boldsymbol{\mathcal{N}}^*_{\boldsymbol{\mathcal{S}}}$, troubled cells $K \in \mathcal{K}^*_h$ and $\gamma \in \mathcal{V}$.

3. **Extrapolate:**

$$\left[\boldsymbol{w}^{K,i}\right]_{d,e,\boldsymbol{\alpha},\gamma} = \left[\hat{\boldsymbol{u}}^{K,i}\right]_{\boldsymbol{\alpha},\gamma} + \frac{e}{2}\left[\hat{\boldsymbol{\delta}}^{K,i}_d\right]_{\boldsymbol{\alpha},\gamma} \tag{2.130}$$

for all subgrid cells $K_{\boldsymbol{\alpha}} \subset K$, $\boldsymbol{\alpha} \in \boldsymbol{\mathcal{N}}^*_{\boldsymbol{\mathcal{S}}}$, troubled cells $K \subset \mathcal{K}^*_h$, $\gamma \in \mathcal{V}$, $d \in \mathcal{D}$ and $e \in \{-1,+1\} := \sigma$.

4. **Evolve:**

$$\begin{aligned}\left[\boldsymbol{w}^{K,i+\frac{1}{2}}\right]_{d,e,\boldsymbol{\alpha},\gamma} = &\left[\boldsymbol{w}^{K,i}\right]_{d,e,\boldsymbol{\alpha},\gamma} + \\ &\frac{\Delta t_i}{2}\sum_{d'\in\mathcal{D}}\sum_{e'\in\sigma}\left(e'\left[\boldsymbol{F}\left(\left[\boldsymbol{w}^{K,i}\right]_{d',e',\boldsymbol{\alpha}}\right)\right]_{\gamma,d'} / \left[\Delta\boldsymbol{x}^{K_{\boldsymbol{\alpha}}}\right]_{d'}\right) + \\ &\frac{\Delta t_i}{2}\left[\hat{\boldsymbol{s}}^{K,i}\right]_{\boldsymbol{\alpha},\gamma} := \\ &\left[\boldsymbol{w}^{K,i}\right]_{d,e,\boldsymbol{\alpha},\gamma} + \frac{\Delta t_i}{2}\left[\hat{\boldsymbol{c}}\right]_{\boldsymbol{\alpha},\gamma}\end{aligned} \tag{2.131}$$

for all subgrid cells $K_{\boldsymbol{\alpha}} \subset K$, $\boldsymbol{\alpha} \in \boldsymbol{\mathcal{N}}^*_{\boldsymbol{\mathcal{S}}}$, troubled cells $K \subset \mathcal{K}^*_h$, $\gamma \in \mathcal{V}$, $d \in \mathcal{D}$ and $e \in \sigma$.

5. **Solve Riemann problems:**

$$\left[\boldsymbol{f}^{K,i}\right]_{d,\boldsymbol{\alpha},\gamma} = \left[\boldsymbol{\mathcal{G}}\left(\left[\boldsymbol{w}^{K,i+\frac{1}{2}}\right]_{d,+1,\boldsymbol{\alpha}-\boldsymbol{e}_d}, \left[\boldsymbol{w}^{K,i+\frac{1}{2}}\right]_{d,-1,\boldsymbol{\alpha}+\boldsymbol{e}_d}, \boldsymbol{e}_d\right)\right]_{\gamma} \tag{2.132}$$

   for all subgrid cells $K_{\boldsymbol{\alpha}} \subset K$, $\boldsymbol{\alpha} \in \boldsymbol{\mathcal{N}}^{*}_{\boldsymbol{\mathcal{S}}}$, troubled cells $K \subset \mathcal{K}^{*}_{h}$, $\gamma \in \mathcal{V}$ and $d \in \mathcal{D}$.

6. **Evolve source:**

$$\left[\hat{\boldsymbol{s}}^{K,i+\frac{1}{2}}\right]_{\boldsymbol{\alpha},\gamma} = \left[\boldsymbol{s}\left(\left[\hat{\boldsymbol{s}}^{K,i}\right]_{\boldsymbol{\alpha}} + \frac{1}{2}\left[\hat{\boldsymbol{c}}\right]_{\boldsymbol{\alpha}}\right)\right]_{\gamma} \tag{2.133}$$

   for all subgrid cells $K_{\boldsymbol{\alpha}} \subset K$, $\boldsymbol{\alpha} \in \boldsymbol{\mathcal{N}}^{*}_{\boldsymbol{\mathcal{S}}}$, troubled cells $K \subset \mathcal{K}^{*}_{h}$ and $\gamma \in \mathcal{V}$.

7. **Update solution:**

$$\begin{aligned} \left[\boldsymbol{p}^{L^K,i+1}\right]_{\boldsymbol{\alpha},\gamma} = \left[\boldsymbol{p}^{L^K,i}\right]_{\boldsymbol{\alpha},\gamma} - & \\ & \Delta t_i \sum_{d\in\mathcal{D}} \left(\left(\left[f^{K,i}\right]_{d,\boldsymbol{\alpha}+\boldsymbol{e}_d,\gamma} - \left[f^{K,i}\right]_{d,\boldsymbol{\alpha},\gamma}\right) / \left[\Delta \boldsymbol{x}^{K_{\boldsymbol{\alpha}}}\right]_{d}\right) + \\ & \Delta t_i \left[\hat{\boldsymbol{s}}^{K,i+\frac{1}{2}}\right]_{\boldsymbol{\alpha},\gamma} \end{aligned} \tag{2.134}$$

   for all subgrid cells $K_{\boldsymbol{\alpha}} \subset K$, $\boldsymbol{\alpha} \in \boldsymbol{\mathcal{N}}^{*}_{\boldsymbol{\mathcal{S}}}$, troubled cells $K \subset \mathcal{K}^{*}_{h}$ and $\gamma \in \mathcal{V}$.

2.2 Profiling and Energy-aware Computing on Modern x86 Systems

2.2.1 On the Importance of Performance Profiling in Software Engineering for High Performance Computing

In a High Performance Computing (HPC) context the standard balance of design goals in Software Engineering is shifted towards maximum application performance. In view of the ever-growing complexity of computer architectures performance profiling has become an inevitable tools that a) provides a baseline reference on the current state of a project, b) helps to prioritize, guide and track progress of optimization efforts and c) allows comparison to other state of the art solutions as well as theoretical optima.

Modern x86 processors are equipped with one or more Performance Monitoring Units (PMU) which in conjunction with a suitable operating system provide means to obtain profiling information directly within an application. This so-called Hardware Performance Monitoring (HPM) interface allows programming of performance event counters that in turn allow computation of metrics that are correlated with good application performance and (more recently) energy efficiency (see [15, 16, 17]). Treibig et al. illustrate best practices on how HPM can be used for performance engineering on modern multi-core processors, offer a taxonomy of common causes for suboptimal application performance and offer guidance on how profiling can be used to identify and tackle them [18].

The rest of this section is organized as follows: We first give a brief summary on the x86 Instruction Set Architecture (ISA) and its importance in current HPC systems. We then focus in more detail on features and limitations of HPM in modern x86 processors and on means how to access them in the context of a normal user-space application. The section is concluded with remarks on on-chip energy monitoring provided by some of the most recent generations of x86 processors and a literature review on findings concerning the accuracy on this feature.

2.2.2 The x86 Instruction Set Architecture and the Current Prevalence of x86 in High Performance Computing

The term x86 refers to a set of Instruction Set Architectures (ISA) based on the philosophy of the Intel 8086 16-bit microprocessor introduced in 1978 [19]. Over the years many additions to the original ISA such as support for 32-bit (1985) and 64-bit (2003) memory addressing or the introduction of special purpose instructions e.g. for vectorized floating-point operations (MMX, SSE, AVX) or encryption (AES-NI) have been made, but nevertheless all of these changes are in theory backward compatible [20]. This means that even though modern x86 processors are 64-bit capable, consist of multiple cores with complex multi-level cache hierarchies and sophisticated out of order execution pipelines, they can still correctly execute the instructions that make of a program originally written for an 8086 [19]. Due to the large number of instructions specified in a modern x86 ISA and due to the comparably complex and sometimes special purpose nature of some of them, x86 is considered to be an instance of the complex instruction set computing (CISC) paradigm (as opposed to RISC, reduced instruction set computing). Throughout this thesis we use the term "modern x86 processor" to denote a 64-bit capable multi-core processor that implements a modern x86 ISA. Even though historically more than ten companies have produced x86 processors, nowadays this means that we refer to a processor manufactured by either Intel and AMD[4].

The famous TOP500 list ranks the word's fastest supercomputers in terms of floating-point performance measured using the LINPACK benchmark. The

[4] The market share of Intel in this "duopoly" is estimated to be between 80% and 98% in the three market segments server, desktop and laptop [21].

Architecture	Count	Percentage
x86	468	93.6%
Power	23	4.6%
SPARC	7	1.4%
Sunway	2	0.4%

(a) CPU Architecture

Accelerator	Count	Percentage
None	406	81.2%
GPU	66	13.2%
Xeon Phi	23	4.6%
Other	5	1.0%

(b) Accelerator Cards

Table 2.1: Distribution of CPU architecture and accelerator cards of the supercomputers listed in the June 2016 Top500 list [22]. Systems that have both GPUs and Xeon Phi accelerator cards are listed as "Other".

list is published biannually in June and December by Strohmaier et al. [22]. The June 2016 edition illustrates the strong prevalence of x86 in HPC (see table 2.1a). 468 of the 500 systems listed at the moment are based x86 processors; 455 of them use CPUS manufactured by Intel.

A major trend directly reflected in the TOP500 list is the growing popularity of dedicated accelerator cards (see table 2.1b). Such devices are usually connected to the main CPU via the PCI Express bus, require additional power supply and sometimes come with their own form of interconnection technology to allow for direct communication that avoids involvement of host memory or CPU. Applications that exhibit a great degree of mostly homogeneous and independent parallelism such as many types of simulations in scientific computing that in terms of runtime are dominated by vectorizable floating-point operations can greatly benefit from the presence of accelerator cards. Figure 2.1 illustrates the growing number of supercomputing systems that employ accelerators over time. Starting approximately in 2010 and 2012, respectively, two types of accelerators can be identified as the main drivers of this trend:

1. Graphics processing units used in the context of general purpose computing (GPGPU), most prominently NVIDIA devices implementing the company's Compute Unified Device Architecture (CUDA).
2. Accelerators based on the many integrated core (MIC) paradigm, that is devices from Intel's Xeon Phi series. Xeon Phi devices in essence consists of a great number (> 30) of small, interconnected x86 processors that support wider vector instructions than contemporary CPUs that are all located on a single chip.

As of June 2016 94 of the 500 machines on the list employ accelerators. Even though more recently the rate of growth in the use of accelerators seems to be slowing down, it is predicted that the demand for accelerator devices specialized in parallel floating-point computations will nevertheless remain strong [23]. This is furthermore underscored by the facto that even though

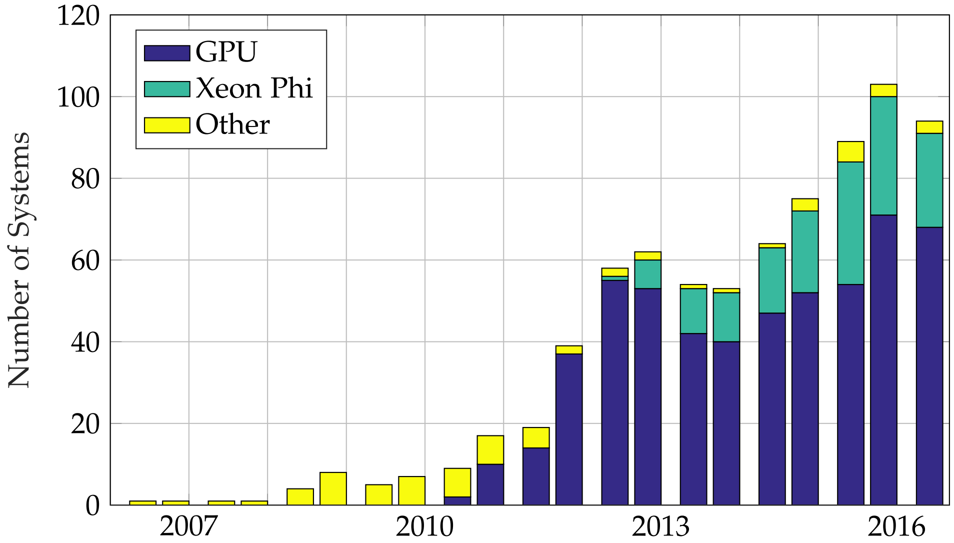


Figure 2.1: Number of supercomputing systems on the TOP500 list [22] that employ accelerator cards between June 2006 (the time when the first such system appeared on the list) and June 2016. Systems that have both GPUs and Xeon Phi accelerators are listed as "Other". The list is updated twice a year in June and November. The bars are centered on the day of release.

only three out of the ten fastest systems today employ accelerators, nine out of the ten most energy-efficient systems on the TOP500 list (consequently leading the so-called Green500 list [24]) do.

The review of the Top500 list above clearly illustrates that at the moment x86 dominates for HPC systems on a broad scale. Considering however trends such as the use of GPUs as special purpose accelerators and the fact that the leading system "Sunway TaihuLight" is based on a custom architecture developed[5] within China might give rise to the presumption that the market of HPC systems will become more diverse within the next couple of years. The four largest supercomputers expected to go in service in the United States by 2018 are either based on NVIDIA GPGPUs (Summit [25] and Sierra [26]) or the third generation of Intel's Xeon Phi product line then to be used directly as a CPU (Theta and Aurora [27]).

2.2.3 Hardware Performance Monitoring in Modern x86 Processors

In 1993 Intel introduced the Pentium 60 and Pentium 66 as the earliest members of the first generation Pentium microprocessor family [28]. From documentation available to the general public it was well known that Pentium

[5] partly in response to export restrictions

processors collected a lot of statistics on the interaction between running code and the hardware. The means how to access the data, however, were left undocumented or required signing a nondisclosure agreement with the manufacturer. Since obtaining the metrics hinted towards in the documentation would be of great value in optimizing application performance, enthusiasts such as Terje Mathisen started to employ reverse engineering techniques and eventually managed to recover the complete set of available performance counter events [29].

Today almost all processors independent of their architecture provide some form of hardware performance monitoring (HPM) for low overhead collect of metrics describing the interaction between code and hardware [30]. The performance monitoring units (PMUs) on modern multi-core processors consist of a small number of (most commonly four) special-purpose registers (often called module specific registers (MSRs)). There is usually one PMU dedicated to each core (more precisely hardware thread if simultaneous multithreading (SMT) is available) and additional PMUs for resources shared between cores such as the memory controller or the interconnection bus. PMU registers can be programmed freely to count the number of occurrences of hardware events such as the retirement of an instruction, a cache miss or a successful branch prediction. If used correctly they provide insight into not only where bottlenecks are in the program, but also help explain the reason why [31]. The special instructions necessary to do so can only be executed with kernel privileges (ring 0) and are not part of the ISA so that for example in the x86 case there is no guaranteed backward compatibility. In Linux there are several kernel modules that allow direct or indirect access to performance counters, most prominently there are OProfile [**?**], perf [**?**] and msr. The de-facto standard library for HPM on Linux is PAPI [**?**], which in turn is built on top of the perf interface. Other libraries such as LIKWID [32] instead make use of msr, a module that maps the MSRs to a device file interface and due to its simplicity promises lower overheads. Documentation on the use of MSRs are available in [20, ch. 19] and [33, ch. 3.2.5] for x86 processors by Intel and AMD, respectively. In principle it would of course be possible for example in a scientific simulation code to directly use the perf or msr kernel interface, but due to poor documentation and the volatile nature of the syntactics and semantics of counter events across the various microprocessor architectures, generations and manufacturers this approach is usually unfeasible in practice.

In general there are two approaches towards employing HPM, namely sampling and probing [30]. The former involves periodically interrupting program execution using system interrupts and to read performance counter values and sampling of the call stack to correlate the obtained data with a section in the program. Overhead and inaccuracy introduced by this procedure is determined by system load and the rate at which interrupts take place. Sampling does not require modification of code or binary, but due to the unavailability of debug information and compiler optimizations such as inlining the mapping of performance data to specific lines in code might be inaccurate. Probing, on the other hand, requires that before and after sections of interest instrumentation code is added. In essence this code programs, resets and starts the performance counters prior to the execution of the section of interest and once it has finished it reads the final counter values and stores them in a data structure ready for further analysis. The overhead of probing might be significant, especially if the part of the program to be profiled is executed very often (see [34] for a practical example). On the other hand probing is considered to be more accurate than sampling, since it does not interrupt code execution (and thereby avoids altering the state of the processor artificially) and the procedure is only minimally invasive with respect to the generated instruction stream, i.e. the only way in which the instrumentation code can have an impact on the section of interest might be compiler optimizations across the boundaries of the section which can not be done any more.

We will now limit our scope to the probing approach applied in the context of modern x86 microprocessors and give an overview on overhead and accuracy of performance counters reported in literature. A great amount of research has been done over the years concerning the overhead of various ways to access performance counters in user-space applications (see for example [30, 35, 36]). Due to its great importance for the profiling infrastructure presented in this thesis later on we cite table 2.2 from [30] measuring the different overheads of reading and writing (i.e. resetting, starting and stopping) a performance counter from within the kernel, a privileged user-space application accessing the msr device file through the LIKWID library ("direct") or an unprivileged user-space application that uses the access daemon provide by LIKWID on an Intel Haswell CPU. The latter scenario is common in the context of data centers where user applications can not run with superuser privileges but still need to use performance counters. The LIKWID daemon, for example, is available on both the SuperMUC and the CoolMUC-2 systems at Leibniz Supercomputing Center. The overhead of libraries based on the perf interface such as PAPI is found to be slightly larger.

Literature on the accuracy of the several hundred (e.g. >400 for Haswell) counters available in modern x86 processors is limited. While metrics such as cycle count are by design accurate down to a single cycle, accuracy varies dramatically for others and error margins might be dramatic in some scenarios. Research presented by Thomas Röhl [37] on Haswell CPUs indicates that common metrics such as branch prediction ratio, instruction retirement, memory bandwidth and load/store ratio haven an average error of less or equal than 0.2%. Performance counters concerned with the multi-level

Mode	Read	Write
Daemon	13144	11388
Direct	1292	656
Kernel	888	312

Table 2.2: Counter access time (median) in cycles for LIKWID on an Intel Xeon E3-1240 v3 CPU (Haswell microarchitecture). Cited from [30].

caching mechanism and its complex non-uniform memory access characteristics are prone to larger error margins; and example would be L1-L2 cache transfer bandwidth with an average error $> 5\%$ in some scenarios. The undocumented counter for advanced vector extensions (AVX) floating-point instructions in general is surprisingly accurate for a range of numerical benchmarks ($< 0.05\%$), especially when compared to results from earlier microprocessor generations. Significant overcounting can happen in presence of certain AVX instructions such as the ones used to copy a subset of the values in a vector register to another. In summary we can conclude that most common performance metrics obtained from the PMUs are highly precise and if used correctly are invaluable in HPM-assisted performance engineering. Most of the time the obtained counter values are accurate and stable enough to allow not just for qualitative but also for quantitative comparison. Since however availability, semantics and accuracy of the metrics depend to a great degree on manufacturer, generation and model of the microprocessor as well as on characteristics of the scenario under consideration (e.g. single-core vs. multi-core), a lot of factors need to be taken into consideration before valid conclusions can we drawn. The overhead introduced by layers of abstraction between counter register and user application can introduce significant overhead compared to a simple register access in certain scenarios. However due to the constant nature of the overhead and the fact that it usually does not dominate the measurement, a well-calibrated profiling instrumentation can be tuned to take overheads into account so that the adjust metrics come remarkably close to the ones that could have been obtained from direct register access.

2.2.4 Energy Monitoring in Modern x86 Processors

The servers in common data centers and to a lesser degree also in supercomputing centers almost never operate at their peak capacity all at the same time. This is the reason why today power supply to such facilities is usually underprovisioned, i.e. it is not designed with respect to the absolute peak, but rather to some worst-case average power consumption [17]. This gives rise to a more recent introduction to x86 processors: On-chip power estimation and capping capabilities.

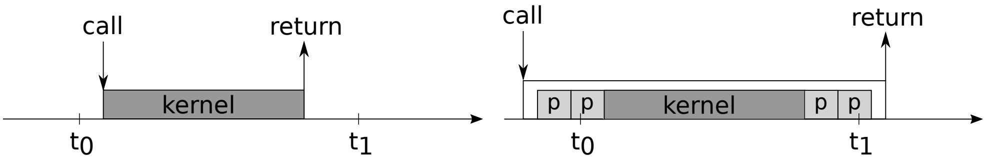
Starting in 2011 with the Sandy Bridge microprocessor architecture Intel introduced the so-called Running Average Power Level (RAPL) interface as a way to prescribe upper bounds on the current power consumption of a CPU [17]. The following year AMD introduced a similar technology called Application Power Management (AMP) [38]. For the resulting embedded control task to be feasible a sophisticated on-chip model to estimate the current power consumption based on architectural events was added [39]. Both technologies expose these estimates in form of model-specific registers that are updated at a rate of about 1kHz (Intel) or 0.1kHz (AMD) [40]. In the case of Intel CPUs these MSRs contain the total amount of consumed energy as a multiple of an architecture dependent base unit (e.g. $61\mu J$ or $15.3\mu J$ for Sandy Bridge, Haswell and Haswell-EP) since the last reset of said counter. The estimates cover so-called power domains, typically separate values for the sum over all cores, for all cores plus shared on-chip resources and for DRAM memory can be obtained. AMD CPUs, on the other hand, provide power estimates again as a multiple of an architecture defined granularity (e.g. $3.8mW$ for the Bulldozer microarchitecture). From now on we will use the terms "RAPL counters" to denote on-chip power estimation capabilities by both vendors.

As mentioned in the introduction energy efficiency is considered to be one of the greatest challenges in designing software for exascale machines. Reliable, low overhead on-chip estimation of CPU and DRAM energy consumption is an invaluable tool in optimizing applications towards this variable and seems to be a favorable alternative in comparison with actual measurement based on complex hardware instrumentation. Before we illustrate how RAPL counters can be used within the profiling infrastructure for ExaHyPE, we first conduct a literature review on accuracy validation, major pitfalls, subsystem modeling (from package to core) and other important observations from practice. Due to a lack of literature on AMD hardware and the fact that most of our test systems as well as those used in literature are based on these architectures, we will mostly focus on Intel Sandy Bridge and Haswell CPUs. We express the hope that accuracy bounds that hold for these architectures will act as lower bounds for future microprocessor generations.

Hackenberg et al. [40] have used external equipment including high-frequency hall effect sensors to measure power consumption at the wall outlet and the mainboard power supply of two Intel Sandy Bridge systems and one AMD Bulldozer system. For a variety of benchmarks occupying all available physical cores for eight seconds they find a small constant offset between externally measured values and those reported by the RAPL interface. The offset is negative in general, i.e. RAPL slightly underestimates the amount of consumed energy; if the DRAM power domain is included the gap closes further and correlation increases. For computationally intensive tasks the offset is slightly lower than for memory-intensive tasks. The results furthermore shows that Sandy Bride reports an almost exact only slightly overestimated value for an idling system; AMP on Bulldozer does not seem to compensate correctly for cores in low power states. The group highlights that measurements close to the counter update rate are challenging due to the variability in the deltas between two consecutive updates and the lack of a precise time stamp that could compensate for it. In a more recent publication [41] Hackenberger et al. repeat their measurements for Haswell-EP and find “a significant improvement compared to RAPL on previous processor generations”. The correlation for the sum of package and DRAM RAPL is “almost perfect” and if in addition a quadratic fit is employed the deviation exceeds 3W at no point. They point out that due to a new RAPL mode for the DRAM domain that is based on actual measurements the reported values are now extremely precise throughout all load scenarios.

Desrochers et al. [42, 43] instrument a Haswell desktop system in a similar way as in [40]: Power supply is intercepted at the wall outlet, at the mainboard connector and due to their special interest in RAPL estimates for DRAM also at the respective memory banks. They again find a small constant offset in the reported RAPL CPU package values for both idle and fully loaded scenarios. If DRAM dims are idle and enter a lower power state RAPL on Haswell (not Haswell-EP) strongly underestimates its power consumption.

Hähnel at el. [44] use a “manually instrumented board” to compare power consumption values reported by the RAPL interface of a Sandy Bridge processor to measurements done with external equipment. They again find “that the curves’ characteristics are identical”, but measure a constant offset they attribute to the fact that they neglect DRAM power consumption altogether. They are particularly concerned with measuring short code paths ($<$ 5ms). Since the RAPL counters are only updated at a rate of 1kHz with jitter of about ±2% this task is particularly challenging. Figure 2.2a illustrates the problem: We can not align the start of our section of interest here denoted by “kernel” with a counter update, in fact we do not even know precisely when this will be the case. Similarly the point in time when the function returns will in general not be aligned with an update so that in this naive approach we would wrongly attribute energy consumption from before and after the function call to the measurement. Figure 2.2b, on the other hand, illustrates a possible solution. Before starting the actual “kernel”, we first repeatedly poll the RAPL MSR and count the number of tries that are need until a change of the counter value is observed. We then execute the kernel and repeat the same procedure once it has finished. Via a priori calibration we can precisely determine the amount of energy that is required for one single poll simply by executing the procedure sequentially for a significant number of times and then computing the mean for a single register



(a) Naive approach

(b) Approach based on repeated counter probing

Figure 2.2: **FIX!!!!** Approaches to measuring power consumption in short code paths using RAPL counters. Illustration adopted from [44].

poll. Since the number of polls we had to wait for the counter updates to happen is known we can correctly compensate for this overhead and adjust the overall energy measurement such that it only reflects the kernel function. In a case study based on single-core video decoding it is shown that if a frame is split into twelve slices and the RAPL energy consumption based on the approach explained above is measured for processing each slice individually, the sum over all slices on average differs by 1.13% compared to a measurement spanning the processing of the complete frame at once [44]. Since in ExaHyPE we will mostly be dealing with short kernel calls close to the minimum temporal resolution of RAPL energy counters we implemented this refined measurement procedure in the profiling infrastructure to be discussed in the next chapter.

With current generation x86 processors RAPL counters can not be used to obtain energy consumption estimates for individual cores. A way to get around this limitation is largely based on research tasked with predicting CPU energy consumption from a time before RAPL counters were introduced: Based on selected performance counters and other information available at runtime such as core temperature and clock speed even relatively simple models were able to make remarkably accurate predictions. If these models are calibrating against RAPL estimates for the complete core they can be used to predict the energy consumption of a single core.

Goel et al. [45]proposed a model based on piecewise linear regression based on temperature and a hardware-specific choice of the four performance counters with highest correlation to predict power fore each core individually. Validation against measurements of total CPU power consumption for various x86 microprocessors introduced between 2007 and 2009 yields a median error of 1.1% - 5.2% for a diverse set of benchmark scenarios.

Also prior to the general availability of RAPL counters Takouna et al. [46] proposed a linear model based on frequency (required to be the same for all cores), squared frequency and number of active cores:

$$P(F, n) = \theta_3 F^2 + \theta_2 F + \theta_1 n + \theta_0. \tag{2.135}$$

Validation against a Nehalem-EP server with build-in CPU power measurement capabilities yields that the prediction error is less than 7% for 95% of the tested combinations of frequency F and active cores n.

Yasin et al. [47] used a particularly simple model based on normalized frequency of the individual cores of the form

$$P(\boldsymbol{c}) = \theta_0 + \theta_1 \sum_{n=1}^{N} \boldsymbol{f}_n, \tag{2.136}$$

where N is the total number of cores in the system and $\boldsymbol{f}_n = \boldsymbol{F}_n / F_{\max}$ is the normalized frequency of the n-th core. For a system consisting of two Sandy Bridge processors and the model calibrated against RAPL power measurements they report a mean absolute error of 4.01% and 5.23% if core power saving states (C-states, P-states) are disabled or enabled, respectively.

Even though literature in the context RAPL counters is admittedly limited and oftentimes methodically unclear or questionable, we can still conclude that the approach of using simple linear models to predict core-level power consumption from per package aggregates is promising. In practice, however, careful calibration and extensive sanity checking is required on a case-by-case basis. Since RAPL counters on Intel platforms report total energy consumption rather than average power intake and due to the jitter in the update rate described above it is justified to assume that estimation of per-core energy consumption will be more accurate than per-core power consumption.

Summing up the key findings of the literature review we can conclude that RAPL counters as the standard way for energy monitoring on modern x86 processors are an important tool when optimizing applications for energy consumption. Under no circumstances do the provided estimates violate the ordering constraints of instruction streams with respect to an ordering based on real measurement with external instrumentation. Then can therefore be used as a cost function to guide manual or automatic optimization. The accuracy of the counters seems to be good enough for most practical use cases; sampling frequencies at the order of magnitude than that at which updates occur are challenging, but possible using the probing approach presented above. Further improvements with respect to temporal granularity in future architectures would nevertheless be highly beneficial. The estimation of power consumption for individual cores on a multi-core processor using simple linear models seems to be feasible and usually results in low relative errors compared to global RAPL measurements.

Chapter 3

A Profiling Infrastructure for ExaHyPE

- General architecture
- Architecture profiling (also likwid, intel pcm)
- Functionality

Likwid ("Like I Knew What I'm Doing"):

- Library that allows manage affinity (discover, pinning), count hardware performance events on x86 Linux
- Builds upon Linux msr kernel module to access model specific registers (MSRs) via a device file interface
- Provide access, nothing more. Groups.
- Lightweight and low-overhead
- Convenience features: logical thread group syntax, pre-configured performance groups
- If regions of interest overlap then overhead influences measurement itself
- Directly accesses MSRs and PCI address space. No mapping to lines, but regions instead (Uncore HPM units: Valid within per chip scope, cannot be mapped to specific cores)
- Keep logic outside of kernel:
  - Support for new hardware does not depend on kernel version
  - Lower hurdle for developers, quicker integration of new features
  - One implementation, single point of failure

- Complicated to choose meaningful events, might be inaccurate. Standardized subsets. Intel: architectural performance monitoring (breaking, bad documentation, too late)
- Implements events and names them according to vendor manuals; uncore: memory controller only; ; additional sources: RAPL for energy, TM/TM" for temperature
- Problem of changing names and capabilities is solved by introduction of so-called performance groups, predefined event sets and derived metrics (custom sets can be defined)
- Modes: Wrapper mode: Events are counted for complete run; timeline mode: sampling; marker api: 5 call api (start, stop, ...), wrapper still needed; marker api: restrict measurements to certain parts; likwid C-API: fine grained control, more complicated, wrapper is based on C-API, standalone, linked into executable.
- Access modes: Direct (root), access daemon (socket file, filtering, access control, broker, security, HPC environment, required for some PCI-based Uncore counters)
- Lower overhead than PAPI

Intel PCM: Fewer features, made by manufacturer.

Implementation:

- Overhead: Hashtable lookup, but since regions of interest do not overlap this is no issue
- Likwid Marker API vs. ExaHyPE profiling
- Pinning: Either using Likwid API or externally at the moment. Will be handled by ExaHyPE in the future.
- Abstraction, so that multiple libraries can be supported: User implementation, confidence, availability, extensibility
- Future: PAPI (builds upon perf kernel interface), MeterPu (cite, based on Intel PCM amongst others)

Chapter 4

Preliminary profiling results, case studies

System: Coolmuc, HW, normal frequency, turbo disabled, HT disabled?

Best practices: Performance patterns

- Load imbalance
- **Bandwidth saturation** (main memory, L3)
- **Strided or erratic data access** ("Cache-based architectures require contiguous data accesses to make efficient use of bandwidth due to the cache line concept". badly ordered loop nests, inappropriate data structure)
- **Bad instruction mix** (compiler, degree of vectorization, expensive operations (sqrt, divide))
- **Limited instruction throughput** (load, multiply), related to previous
- **Microarchitectural anomalies**: Alignment, ...
- Synchronization overhead
- False cache line sharing
- **Bad page placement on ccNUMA** (pinning)

HPM groups:

- Memory bandwidth close to peak
- Low bandwidth, high store/load counts, program memory bound, cache ratios
- Ratio FP / instructions retired
- Static analysis: High pressure on single unit

- Hardware specific alignment things
- System vs. user time?!
- Analytic benchmark: Introduction, derivation
- Pie-chart per kernel
- Case-studies HPM-based optimization efforts: Cache-misses, compile-time (→ Toolkit philosophy)
- Degree → Wallclock, Energy (AMR)
- Static mesh Δx → Error for polynomials (convergence tables)

Chapter 5

Conclusion and Outlook

- PA is important
- ExaHyPE as an answer to exascale challenges
- Applications

Chapter 6

Acknowledgment

Appendix A

Computation of the Discrete ADER-DG Operators

In [1]:
```
import numpy as np
np.set_printoptions(precision=3);
import sympy as sp
sp.init_printing(use_latex=True);
x = sp.symbols("x");

## Settings
N = 4;          # Degree of the ADER-DG scheme in space and time
N_S = 2*N + 1;  # Number of subcell averages on the fine grid
##
```

Legendre Polynomials

$P_0(x) = 1$, $P_1(x) = x$, $P_{n+1}(x) = \frac{1}{n+1}[(2n+1)xP_n(x) - nP_{n-1}(x)]$

In [2]:
```
def NextLegendrePolynomial(n, P_, P__):
    P = ((2*n + 1) * x * P_ - n * P__) / (n+1);
    return P.simplify();

P = []; P.append(1); P.append(x);

for i in range(1, N+1):
    P.append(NextLegendrePolynomial(i, P[i], P[i-1]));

P = P[N+1];
P
```

Out[2]:
$$\frac{x}{8}\left(63x^4 - 70x^2 + 15\right)$$

Gauss-Legendre nodes $\tilde{\xi}$

In [3]:
```
xi_tilde_sym = sorted(sp.solve(P, x));
xi_tilde_sym
```

Out[3]:
$$\left[-\sqrt{\frac{2\sqrt{70}}{63} + \frac{5}{9}}, \quad -\sqrt{-\frac{2\sqrt{70}}{63} + \frac{5}{9}}, \quad 0, \quad \sqrt{-\frac{2\sqrt{70}}{63} + \frac{5}{9}}, \quad \sqrt{\frac{2\sqrt{70}}{63} + \frac{5}{9}}\right]$$

Gauss-Legendre nodes $\hat{\xi}$ on $[0, 1]$

In [4]:
```
xi_hat_sym = [((xi+1) / 2).simplify() for xi in xi_tilde_sym];
xi_hat = np.array([xi.evalf() for xi in xi_hat_sym]).astype(np.float64);
print xi_hat;
```

```
[ 0.047  0.231  0.5    0.769  0.953]
```

Gauss-Legendre weights $\hat{\boldsymbol{\omega}}$ on $[0, 1]$

In [5]:

```
A = np.matrix([xi_hat**i for i in range(0, N+1)]);
b = np.array([sp.integrate(x**i, (x, (0, 1))).evalf()
              for i in range(0, N+1)]).astype(np.float64);

omega_hat = np.linalg.solve(A, b);
print omega_hat;
```

```
[ 0.118  0.239  0.284  0.239  0.118]
```

Lagrange interpolation polynomials L_i on $[0, 1]$ with nodes $\hat{\boldsymbol{\xi}}$

$L_i(\xi) = \prod_{j \neq i} \frac{\xi - \hat{\boldsymbol{\xi}}_j}{\hat{\boldsymbol{\xi}}_i - \hat{\boldsymbol{\xi}}_j}, i = 0, \ldots, N$

In [6]:

```
def L(i, xi_hat):
    xi_skip = xi_hat[:i] + xi_hat[i+1:];
    numerator = sp.prod([x - xi for xi in xi_skip])
    denominator = sp.prod([xi_hat[i] - xi for xi in xi_skip])
    return numerator / denominator

psi = [sp.lambdify(x, L(i, xi_hat_sym)) for i in range(N+1)]
```

In [7]:

```
def dL(i, xi_hat):
    return sp.diff(L(i, xi_hat))

dpsi = [sp.lambdify(x, dL(i, xi_hat_sym)) for i in range(N+1)]
```

Left Reference Element Flux Operator $\boldsymbol{l}$

In [8]:

```
l = np.array([psi[i](0.0) for i in range(0, N+1)]);
print l
```

```
[ 1.551 -0.893  0.533 -0.268  0.076]
```

Right Reference Element Flux Operator $\boldsymbol{r}$

In [9]:

```
r = np.array([psi[i](1.0) for i in range(0, N+1)]);
print r
```

```
[ 0.076 -0.268  0.533 -0.893  1.551]
```

Right Reference Element Mass Operator $\boldsymbol{R}$

In [10]:
```
R = np.fromfunction(np.vectorize(
        lambda i, j: psi[i](1.0) * psi[j](1.0)), (N+1, N+1), dtype=int);
print R
```

```
[[ 0.006 -0.02   0.041 -0.068  0.118]
 [-0.02   0.072 -0.143  0.239 -0.416]
 [ 0.041 -0.143  0.284 -0.476  0.827]
 [-0.068  0.239 -0.476  0.798 -1.386]
 [ 0.118 -0.416  0.827 -1.386  2.407]]
```

Reference Element Stiffness Operator $\boldsymbol{K}$

In [11]:
```
K = np.fromfunction(np.vectorize(
        lambda i, j: omega_hat[j] * dpsi[i](xi_hat[j])), (N+1, N+1),
                    dtype=int);
print K
```

```
[[-1.201 -0.46   0.171 -0.117  0.131]
 [ 1.825 -0.363 -0.817  0.444 -0.464]
 [-0.958  1.15   0.    -1.15   0.958]
 [ 0.464 -0.444  0.817  0.363 -1.825]
 [-0.131  0.117 -0.171  0.46   1.201]]
```

Iteration Matrix $\tilde{\boldsymbol{K}}$

In [12]:
```
Ktilde = np.linalg.inv(R - K);
print Ktilde
```

```
[[ 0.53  -0.124  0.086 -0.066  0.044]
 [ 1.039  0.559 -0.151  0.102 -0.066]
 [ 1.007  1.022  0.57  -0.151  0.086]
 [ 0.981  1.016  1.022  0.559 -0.124]
 [ 1.017  0.981  1.007  1.039  0.53 ]]
```

Projection Operator P

In [13]:

```
P = np.fromfunction(np.vectorize(
        lambda i, j:
        sum([omega_hat[k] *
             psi[j](1.0/N_S * i + 1.0/N_S * xi_hat[k])
             for k in range(0, N+1)])), (N_S, N+1), dtype=int);
print P
```

```
[[  9.472e-01   5.620e-02  -2.341e-03  -1.894e-03   8.191e-04]
 [  2.055e-01   9.434e-01  -2.174e-01   9.415e-02  -2.558e-02]
 [ -5.363e-02   8.698e-01   2.435e-01  -7.997e-02   2.030e-02]
 [ -6.623e-02   4.256e-01   7.658e-01  -1.644e-01   3.913e-02]
 [ -1.333e-03   1.087e-02   9.809e-01   1.087e-02  -1.333e-03]
 [  3.913e-02  -1.644e-01   7.658e-01   4.256e-01  -6.623e-02]
 [  2.030e-02  -7.997e-02   2.435e-01   8.698e-01  -5.363e-02]
 [ -2.558e-02   9.415e-02  -2.174e-01   9.434e-01   2.055e-01]
 [  8.191e-04  -1.894e-03  -2.341e-03   5.620e-02   9.472e-01]]
```

Reconstruction Operator R

In [14]:

```
m1 = np.append(2*P.T.dot(P), [omega_hat], axis=0);
m1 = np.append(m1, np.append(omega_hat, 0.0).reshape(N+2, 1), axis = 1)

m2 = np.append(2*P.T, [1.0/N_S * np.ones(N_S)], axis=0)

Rtilde = np.linalg.solve(m1, m2);
R = np.delete(Rtilde, N+1, 0)
print R
```

```
[[ 1.014  0.113 -0.13  -0.073  0.04   0.074  0.007 -0.075  0.03 ]
 [-0.064  0.514  0.468  0.195 -0.041 -0.112 -0.021  0.095 -0.034]
 [ 0.038 -0.137  0.066  0.32   0.426  0.32   0.066 -0.137  0.038]
 [-0.034  0.095 -0.021 -0.112 -0.041  0.195  0.468  0.514 -0.064]
 [ 0.03  -0.075  0.007  0.074  0.04  -0.073 -0.13   0.113  1.014]]
```

Bibliography

[1] Lewis Fry Richardson. *Weather Prediction by Numerical Process*. Cambridge University Press, 2007.

[2] Eleuterio F. Toro. *Riemann Solvers and Numerical Methods for Fluid Dynamics*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[3] Michael Dumbser, Olindo Zanotti, Raphael Loubere, and Steven Diot. A Posteriori Subcell Limiting of the Discontinuous Galerkin Finite Element Method for Hyperbolic Conservation Laws. *Journal of Computational Physics*, 278:47–75, December 2014.

[4] Bernardo Cockburn and Chi-Wang Shu. TVB Runge-Kutta local projection discontinuous Galerkin finite element method for conservation laws. II. General framework. *Mathematics of computation*, 52(186):411–435, 1989.

[5] Bernardo Cockburn, George E. Karniadakis, and Chi-Wang Shu, editors. *Discontinuous Galerkin Methods: Theory, Computation and Applications*. Springer, Berlin; New York, softcover reprint of the original 1st ed. 2000 edition edition, January 2000.

[6] Dominic Etienne Charrier. ADER-DG on adaptive spacetrees, 2016, July 6.

[7] Michael Dumbser, Cedric Enaux, and Eleuterio F. Toro. Finite volume schemes of very high order of accuracy for stiff hyperbolic balance laws. *Journal of Computational Physics*, 227(8):3971–4001, April 2008.

[8] W. Michael Lai, David H. Rubin, David Rubin, and Erhard Krempl. *Introduction to Continuum Mechanics*. Butterworth-Heinemann, 2009.

[9] Søren Hauberg John W. Eaton, David Bateman and Rik Wehbring. *GNU Octave Version 4.0.0 Manual: A High-Level Interactive Language for Numerical Computations*. 2015.

[10] Michael Dumbser. *Arbitrary High Order Schemes for the Solution of Hyperbolic Conservation Laws in Complex Domains*. Shaker, 2005.

[11] Olindo Zanotti, Francesco Fambri, and Michael Dumbser. Solving the relativistic magnetohydrodynamics equations with ADER discontinuous Galerkin methods, a posteriori subcell limiting and adaptive mesh refinement. *Monthly Notices of the Royal Astronomical Society*, 452(3):3010–3029, 2015.

[12] Sergei Konstantinovich Godunov. A difference method for numerical calculation of discontinuous solutions of the equations of hydrodynamics. *Matematicheskii Sbornik*, 89(3):271–306, 1959.

[13] C. Lawson and R. Hanson. *Solving Least Squares Problems*. Classics in Applied Mathematics. Society for Industrial and Applied Mathematics, January 1995.

[14] Michael Dumbser and Martin Käser. Arbitrary high order non-oscillatory finite volume schemes on unstructured meshes for linear hyperbolic systems. *Journal of Computational Physics*, 221(2):693–723, February 2007.

[15] Bob Steigerwald, Chris; Lucero, Chakravarthy Akella, and Abhishek R. Agrawal. *Energy Aware Computing: Powerful Approaches for Green System Design*. Intel Press, Hillsboro, Or., March 2012.

[16] Shajulin Benedict. Energy-aware performance analysis methodologies for HPC architectures—An exploratory study. *Journal of Network and Computer Applications*, 35(6):1709–1719, November 2012.

[17] H. David, E. Gorbatov, U. R. Hanebutte, R. Khanna, and C. Le. RAPL: Memory power estimation and capping. In *2010 ACM/IEEE International Symposium on Low-Power Electronics and Design (ISLPED)*, pages 189–194, August 2010.

[18] Jan Treibig, Georg Hager, and Gerhard Wellein. Best practices for HPM-assisted performance engineering on modern multicore processors. *arXiv:1206.3738 [cs]*, 7640:451–460, 2013.

[19] Lyla B. Das. *The x86 Microprocessors: 8086 to Pentium, Multicores, Atom and the 8051 Microcontroller, 2nd Edition*. Pearson India, 2 edition, May 2014.

[20] Intel Corporation. *Combined Volume Set of Intel®64 and IA-32 Architectures Software Developer's Manual*. Number 253665-059US. June 2016.

[21] Roger Kay. Intel And AMD: The Juggernaut Vs. The Squid. `http://www.forbes.com/sites/rogerkay/2014/11/25/intel-and-amd-the-juggernaut-vs-the-squid/`, November 2014.

[22] Erich Strohmaier, Jack Dongarra, Horst Simon, and Martin Meuer. TOP500 Supercomputer Sites (June 2016). `https://www.top500.org/lists/2016/06/`, June 2016.

[23] M. Véstias and H. Neto. Trends of CPU, GPU and FPGA for high-performance computing. In *2014 24th International Conference on Field Programmable Logic and Applications (FPL)*, pages 1–6, September 2014.

[24] Wu-chun Feng and Tom Scogland. GREEN500 Supercomputer Sites. `https://www.top500.org/green500/lists/2016/06/`, June 2016.

[25] Summit. Scale new heights. Discover new solutions. `http://www.olcf.ornl.gov/summit/`.

[26] Sierra Advanced Technology System. `http://computation.llnl.gov/computers/sierra-advanced-technology-system`.

[27] Aurora. `http://aurora.alcf.anl.gov/`.

[28] Intel Timeline: A History of Innovation. `http://www.intel.com/content/www/us/en/history/historic-timeline.html`.

[29] Terje Mathisen. Pentium Secrets. *Byte magazine*, October 1999.

[30] T. Röehl, J. Treibig, G. Hager, and G. Wellein. Overhead Analysis of Performance Counter Measurements. In *2014 43rd International Conference on Parallel Processing Workshops*, pages 176–185, September 2014.

[31] Kevin S. London, Jack Dongarra, Shirley Moore, Philip Mucci, Keith Seymour, and Thomas Spencer. End-user Tools for Application Performance Analysis Using Hardware Counters. In *ISCA PDCS*, pages 460–465, 2001.

[32] Jan Treibig, Georg Hager, and Gerhard Wellein. LIKWID: Lightweight Performance Tools. *arXiv:1104.4874 [cs]*, pages 207–216, September 2010.

[33] Advanced Micro Devices Inc. AMD64 architecture programmer's manual volume 2: System programming. 2016.

[34] Bernd Mohr, Darryl Brown, and Allen Malony. TAU: A portable parallel program analysis environment for pC++. In *Parallel Processing: CONPAR 94—VAPP VI*, pages 29–40. Springer, 1994.

[35] Philip J Mucci, Shirley Browne, Christine Deane, and George Ho. PAPI: A portable interface to hardware performance counters. In *Proceedings of the Department of Defense HPCMP Users Group Conference*, pages 7–10, 1999.

[36] V. M. Weaver, M. Johnson, K. Kasichayanula, J. Ralph, P. Luszczek, D. Terpstra, and S. Moore. Measuring Energy and Power with PAPI. In *2012 41st International Conference on Parallel Processing Workshops*, pages 262–268, September 2012.

[37] Thomas Röhl. Performance monitoring on Intel Haswell platforms, October 2015.

[38] Advanced Micro Devices, Inc. AMD Opteron 6200 Series Processors Linux Tuning Guide, 2012.

[39] E. Rotem, A. Naveh, A. Ananthakrishnan, E. Weissmann, and D. Rajwan. Power-Management Architecture of the Intel Microarchitecture Code-Named Sandy Bridge. *IEEE Micro*, 32(2):20–27, March 2012.

[40] D. Hackenberg, T. Ilsche, R. Schöne, D. Molka, M. Schmidt, and W. E. Nagel. Power measurement techniques on standard compute nodes: A quantitative comparison. In *2013 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 194–204, April 2013.

[41] Daniel Hackenberg, Robert Schöne, Thomas Ilsche, Daniel Molka, Joseph Schuchart, and Robin Geyer. An energy efficiency feature survey of the intel haswell processor. In *Parallel and Distributed Processing Symposium Workshop (IPDPSW), 2015 IEEE International*, pages 896–904. IEEE, 2015.

[42] Spencer Desrochers, Chad Paradis, and Vincent M. Weaver. Initial Validation of DRAM and GPU RAPL Power Measurements. Tech report, UMaine VMW Group, August 2015.

[43] Spencer Desrochers, Chad Paradis, and Vincent M. Weaver. A Validation of DRAM RAPL Power Measurements. 2016.

[44] Marcus Hähnel, Björn Döbel, Marcus Völp, and Hermann Härtig. Measuring Energy Consumption for Short Code Paths Using RAPL. *SIGMETRICS Perform. Eval. Rev.*, 40(3):13–17, January 2012.

[45] B. Goel, S. A. McKee, R. Gioiosa, K. Singh, M. Bhadauria, and M. Cesati. Portable, scalable, per-core power estimation for intelligent resource management. In *Green Computing Conference, 2010 International*, pages 135–146, August 2010.

[46] Ibrahim Takouna, Wesam Dawoud, and Christoph Meinel. Accurate mutlicore processor power models for power-aware resource management. In *Dependable, Autonomic and Secure Computing (DASC), 2011 IEEE Ninth International Conference on*, pages 419–426. IEEE, 2011.

[47] M. Yasin, A. Shahrour, and I. A. Elfadel. Ultra compact, quadratic power proxies for multi-core processors. In *2013 IEEE 20th International Conference on Electronics, Circuits, and Systems (ICECS)*, pages 954–957, December 2013.