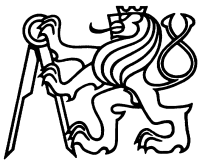
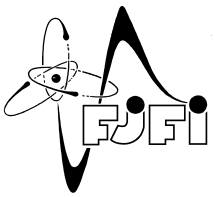


České vysoké učení technické v Praze
Fakulta jaderná a fyzikálně inženýrská



Matematické modelování dvoufázového kompozičního proudění v porézním prostředí

Mathematical modelling of two-phase compositional flow in porous media

Diplomová práce

Autor: **Bc. Jakub Klinkovský**

Vedoucí práce: **Ing. Radek Fučík, Ph.D.**

Akademický rok: 2016/2017

- Zadání práce -

Poděkování:

V prvé řadě děkuji svému školiteli Ing. Radku Fučíkovi, Ph.D. za pečlivost, ochotu a vstřícnost při vedení mé diplomové práce i předchozích prací v rámci studia. Dále děkuji svému konzultantovi Ing. Tomáši Oberhuberovi, Ph.D. za umožnění spolupráce na vývoji numerické knihovny `TNL` a neocenitelné rady při implementaci nestrukturovaných sítí na GPU.

Tato práce vznikla za podpory následujících projektů:

- grant Studentské grantové soutěže ČVUT č. SGS14/206/OHK4/3T/14 s názvem „Pokročilé superpočítačové metody pro matematické modelování přírodních procesů“,
- grant Studentské grantové soutěže ČVUT č. SGS17/194/OHK4/3T/14 s názvem „Aplikace pokročilých superpočítačových metod v matematickém modelování přírodních procesů“,
- grant Ministerstva zdravotnictví České republiky č. NV15-27178A s názvem „Kvantitativní mapování myokardu a dynamiky proudění pomocí MR zobrazování u pacientů s neischemickým postižením srdce – rozvoj metodiky“.

Čestné prohlášení:

Prohlašuji, že jsem tuto práci vypracoval samostatně a uvedl jsem všechnu použitou literaturu.

V Praze dne 3. května 2017 Jakub Klinkovský

Název práce:	**Matematické modelování dvoufázového kompozičního proudění v porézním prostředí**
Autor:	Bc. Jakub Klinkovský
Obor:	Matematické inženýrství
Druh práce:	Diplomová práce
Vedoucí práce:	Ing. Radek Fučík, Ph.D., Katedra matematiky, Fakulta jaderná a fyzikálně inženýrská, České vysoké učení technické v Praze

Abstrakt:

Tato práce se zabývá matematickým modelováním dvoufázového kompozičního proudění v porézním prostředí a efektivním řešením výsledného modelu pomocí pokročilých numerických metod a moderních paralelních výpočetních architektur. Nejprve je popsán fyzikální kontext a matematický model problematiky dvoufázového proudění v porézním prostředí. Dále je sestaveno numerické schéma pro obecnou soustavu n parciálních diferenciálních rovnic s obecnými koeficienty, jejichž volbou lze popsat mnoho konkrétních úloh z dané problematiky. Schéma je založeno na kombinaci hybridní metody smíšených konečných prvků a nespojité Galerkinovy metody pro prostorovou diskretizaci, časová diskretizace je provedena pomocí Eulerovy metody a pomocí semi-implicitního přístupu metody zamrzlých koeficientů je schéma linearizováno v čase. Schéma je odvozeno v obecné prostorové dimenzi a je použitelné i pro nestrukturované diskretizace prostorové oblasti, v této práci jsou použity strukturované sítě tvořené obdélníky ve 2D a kvádry ve 3D a nestrukturované sítě tvořené trojúhelníky ve 2D a čtyřstěny ve 3D. Pro stabilizaci schématu je použita technika upwind a v případě obdélníkových a kvádrových elementů také technika mass-lumping. V další části práce je popsána implementace numerického schématu pro moderní výpočetní architektury GPU a vícejádrových CPU. Řešič je implementován v jazyce C++ s využitím šablonové knihovny `TNL` a paralelizace je provedena pomocí platformy CUDA v případě GPU a OpenMP v případě CPU. Je zde podrobně popsána paralelizace významných kroků algoritmu pro výpočet numerického řešení, jako například sestavení řídké matice a následné řešení soustavy lineárních algebraických rovnic. Při popisu je kladen důraz na klíčové aspekty efektivní implementace, zejména důležitost vhodného rozložení dat v paměti a vliv očíslování elementů nestrukturované sítě na efektivitu výpočtu. Pro zlepšení efektivity výpočtu řešení soustavy lineárních algebraických rovnic pomocí iterační metody GMRES na GPU je popsána modifikace této metody využívající ortogonalizaci pomocí Householderových transformací a kompaktní WY reprezentace. V poslední části práce věnující se numerickým simulacím jsou popsány úlohy nemísivého dvoufázového proudění a kompozičního dvoufázového proudění se známými přesnými řešeními, které byly použity k verifikaci numerického schématu. Konvergence numerického schématu je ověřena pomocí analýzy experimentálního řádu konvergence pro strukturované a nestrukturované sítě ve 2D a 3D. Pro obě úlohy a všechny typy sítí je dále provedena analýza efektivity paralelního výpočtu na GPU a vícejádrovém CPU.

Klíčová slova: dvoufázové proudění, kompoziční proudění, hybridní metoda smíšených konečných prvků, upwind, metoda GMRES, paralelní implementace na GPU, nestrukturované sítě

Title: **Mathematical modelling of two-phase compositional flow in porous media**

Author: Bc. Jakub Klinkovský

Abstract:

This thesis deals with mathematical modelling of two-phase compositional flow in porous media and an efficient solution of the resulting model using advanced numerical methods for modern parallel computational architectures. Firstly, we describe the physical context and the mathematical model of two-phase flow in porous media. Then, we present a multidimensional numerical scheme for a general system of n partial differential equations in a general coefficient form, which can be used to describe many specific problems depending on the choice of the coefficients. The scheme is based on the combination of the mixed-hybrid finite element and discontinuous Galerkin methods for the spatial discretization, the temporal discretization is performed using the Euler method and the semi-implicit approach of the frozen coefficients method is used to linearize the scheme in time. The scheme can work with both structured and unstructured meshes. In this work, we use structured meshes consisting of rectangles in 2D and cuboids in 3D and unstructured meshes consisting of triangles in 2D and tetrahedrons in 3D. The scheme is stabilized by the upwind technique and in the case of rectangular and cuboidal elements, we also use the mass-lumping technique. The next part of the work deals with the implementation of the numerical scheme for modern parallel architectures of GPUs and multicore CPUs. The solver is implemented in the C++ language with the help of the `TNL` library, the CUDA framework for parallelization on GPUs and OpenMP for parallelization on CPUs. A detailed description of notable steps of the computational algorithm, such as the assembly of the sparse matrix and the solution of the linear system, is presented. We focus on the key aspects of the efficient implementation for GPU, notably the importance of proper data layout in memory and the influence of unstructured mesh ordering on the computational efficiency. To improve the efficiency of the linear system solver on GPU, we present a modification of the GMRES method using the orthogonalization by Householder transformations and the compact WY representation. In the last part of the work, we describe benchmark problems for two-phase and two-phase compositional flow in porous media and use their known exact solutions to verify the convergence of the numerical scheme by means of the analysis of the experimental order of convergence. Both benchmark problems are computed on structured and unstructured meshes in 2D and 3D and an efficiency analysis of the parallel computation on GPU and multicore CPU for all mesh types is performed.

Key words: two-phase flow, compositional flow, mixed-hybrid finite element method, upwind, GMRES method, parallel implementation on GPU, unstructured meshes

Obsah

Úvod

Problematika studia procesů v porézním prostředí je velmi bohatá a matematické modelování proudění je jedním ze základních prostředků analýzy a predikce těchto procesů. Matematické modely slouží jako podklad v řadě praktických aplikací, od průmyslových aplikací jako těžba ropy, ukládání CO_2 nebo radioaktivních odpadů, přes ekologické aplikace jako např. ochrana zdrojů pitné vody a simulace šíření kontaminace v podzemí, medicínské aplikace jako např. perfuze v myokardu, až po armádní aplikace jako např. detekce min. Pro studium proudění v porézním prostředí existuje mnoho numerických schémat založených na metodách konečných diferencí, konečných objemů nebo konečných prvků, přičemž každá z metod má své výhody a nevýhody. V této práci popíšeme pokročilé numerické schéma založené na hybridní metodě smíšených konečných prvků a semi-implicitním přístupu k časové diskretizaci, které má řadu výhod: je lokálně konzervativní, umožňuje použití obecných nestrukturovaných sítí, vede na soustavu s pozitivně definitní maticí a schéma lze dobře paralelizovat. Linearizace v čase pomocí metody zamrzlých koeficientů vedoucí na semi-implicitní schéma má oproti plně implicitním metodám výhody v lepší efektivitě výpočtu [39]. V neposlední řadě popíšeme modifikaci hybridní metody smíšených konečných prvků inspirovanou pracemi [17, 18, 25] a představenou v [16], která umožňuje řešit úlohy s degenerující nebo nulovou difúzí.

Numerické simulace komplexních úloh z oblasti modelování proudění vyžadují obrovský výpočetní výkon moderních počítačů. Výpočetní výkon samostatné výpočetní jednotky neomezeně navyšovat kvůli fyzikálním omezením při konstrukci integrovaných obvodů a proto je nutné využít paralelní architektury pro zrychlení výpočtu. Tradiční architektury využívající vícejádrová CPU jsou omezena zejména malou datovou propustností sdílené operační paměti [49]. V posledních letech se rapidně rozvíjí oblast obecných výpočtů na GPU, což jsou výpočetní akcelerátory původně určené pro výpočty spojené s počítačovou grafikou. Architektura GPU adresuje výše uvedené problémy pomocí masivní paralelizace a speciálního typu paměti s řádově vyšší datovou propustností oproti běžné operační paměti. Dnešní GPU disponují typicky tisíci výpočetními jednotkami, které jsou oproti moderním sofistikovaným CPU podstatně primitivnější, ale jako celek poskytují mnohonásobně vyšší výpočetní výkon. Efektivní využití akcelerátorů GPU typicky vyžaduje návrh algoritmů pro danou architekturu, aby bylo zajištěné dobré rozprostření řešené úlohy mezi všechny výpočetní jednotky a efektivní využití hierarchie pamětí.

Cílem této práce je formulace matematického modelu dvoufázového kompozičního proudění v porézním prostředí, popis numerického schématu založeného na hybridní metodě smíšených konečných prvků pro řešení příslušné nestacionární úlohy a masivně paralelní implementace metody na nestrukturovaných sítí. Výsledný program zobecňuje řešič dvoufázového nemísivého proudění [28] vyvinutý autorem v předchozích letech pro jednorozměrné a dvojrozměrné strukturované sítě. Řešič je určen pro paralelní architektury GPU a vícejádrových CPU se sdílenou pamětí a je implementován v jazyce C++ pomocí knihovny `TNL` a platformy CUDA. Součástí práce je popis klíčových aspektů efektivní implementace řešiče, včetně modifikace metody GMRES pomocí Householderových transformací a kompaktní WY reprezentace [27, 44, 50], která umožňuje výrazně lepší efektivitu výpočtu na GPU. Správnost implementace a konvergenci numerického schématu ověříme pomocí testovacích úloh ve 2D a 3D se známým přesným řešením. Dále porovnáme efektivitu paralelního výpočtu na vícejádrovém CPU a urychlení výpočtu na GPU oproti CPU při použití strukturovaných a nestrukturovaných sítí ve 2D i 3D.

Struktura práce

V první kapitole popíšeme danou problematiku ve fyzikálním kontextu, zavedeme základní pojmy nezbytné pro popis porézního prostředí a vícefázového proudění ovlivněného kapilaritou a zformulujeme soustavu parciálních diferenciálních rovnic popisujících kompoziční dvoufázové proudění v porézním prostředí.

Druhá kapitola představuje mezistupeň mezi konkrétními formulacemi úloh kompozičního proudění v porézním prostředí a popisem numerického schématu. Je zde popsána formulace zkoumaných úloh pomocí obecné soustavy n parciálních diferenciálních rovnic s obecnými koeficienty doplněná o počáteční a okrajové podmínky. Dále je uvedena konkrétní volba nelineárních koeficientů pro zkoumané úlohy.

Třetí kapitola se věnuje popisu numerického schématu založeného na kombinaci hybridní metody smíšených konečných prvků a nespojité Galerkinovy metody pro prostorovou diskretizaci, Eulerovy metody pro časovou diskretizaci a semi-implicitním přístupu metody zamrzlých koeficientů pro linearizaci diskrétního schématu v čase. Dále je popsána stabilizace schématu pomocí metody upwind a kapitola je uzavřena popisem algoritmu pro výpočet numerického řešení. Výpočet prvků lokálních matic pro jednotlivé typy elementů včetně použití metody mass-lumping pro úsečky, obdélníky a kvádry je uveden v příloze na konci práce.

Ve čtvrté kapitole jsou popsány klíčové aspekty efektivní implementace numerického schématu pro paralelní architektury GPU a vícejádrových CPU. Nejprve je předvedena důležitost rozložení dat v paměti s ohledem na danou výpočetní architekturu. Poté je popsána paralelizace metody GMRES pro řešení soustav lineárních algebraických rovnic a její modifikace, která umožňuje výrazně lepší efektivitu výpočtu na GPU. Na závěr je popsána implementace nestrukturovaných sítí v rámci knihovny `TNL` a je předvedena důležitost vhodného očíslování entit sítě.

V poslední kapitole jsou prezentovány výsledky numerických simulací. Pro ověření správnosti implementace a konvergence numerického schématu jsou využity testovací úlohy se známým semi-analytickým řešením. Pro úlohy ve 2D a 3D je provedena experimentální analýza řádu konvergence a porovnána efektivita paralelního výpočtu na vícejádrovém CPU a urychlení výpočtu na GPU oproti CPU. Úlohy jsou řešeny s využitím strukturovaných i nestrukturovaných sítě.

Kapitola 1

Dvoufázové kompoziční proudění v porézním prostředí

V první kapitole popíšeme fyzikální kontext a matematický model problematiky dvoufázového proudění v porézním prostředí. Postupně zavedeme základní pojmy potřebné pro popis porézního prostředí a jeho vlastností, veličiny potřebné pro popis vícefázového proudění a transport komponent a uvedeme rovnice popisující toto proudění. V závěru kapitoly zformulujeme soustavy parciálních diferenciálních rovnic pro úlohy, kterými se budeme zabývat v dalších kapitolách.

Definice pojmů jsou převzaty z [2] a [14], kde lze najít podrobnější popis této problematiky.

1.1 Porézní prostředí

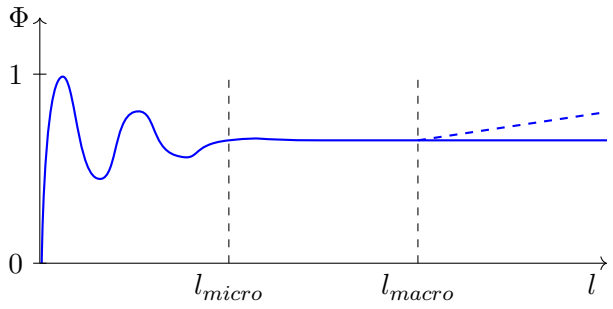
Porézním prostředím rozumíme takové prostředí, které je tvořené primárně pevnými látkami (*skeletem*), ale které obsahuje prázdné dutiny, neboli *póry*. Typickými příklady porézního prostředí jsou zemina složená ze štěrku, písku, jílu a organické hmoty, molitan, nebo rostlinná a živočišná tkáň. Póry mohou být vyplněny jednou nebo více tekutinami (např. vodou, vzduchem), přičemž předpokládáme následující vlastnosti [2]:

- póry tvoří propojenou síť,
- póry jsou dostatečně velké ve srovnání s rozměry molekul tekutin vyskytujících se v systému,
- póry jsou dostatečně malé na to, aby proudění tekutiny bylo řízeno adhezními silami na rozhraní tekutiny a skeletu.

Při splnění těchto předpokladů lze uvažovat matematický model proudění tekutin v porézním prostředí popsaný v této práci.

Na úlohy proudění v porézním prostředí můžeme nahlížet z různých měřítek. V *makroskopickém měřítku* pozorujeme jen některé vlastnosti porézního prostředí, např. rozdílnou průměrnou hrubost zrn písku, ale nejsme schopni rozlišit vnitřní strukturu materiálu. V *mikroskopickém měřítku* jsme schopni rozeznat jednotlivé póry, přičemž tekutiny stále považujeme za kontinuum. Teprve v *molekulárním měřítku* již můžeme pozorovat jednotlivé molekuly tekutin vyplňujících póry. Proudění tekutin lze popsat v mikroskopickém měřítku pomocí soustavy Navierových–Stokesových rovnic s příslušnými okrajovými podmínkami. Porézní prostředí má ale obvykle složité geometrické uspořádání, které není známo na mikroskopické úrovni, navíc okrajových podmínek by bylo příliš mnoho na to, aby bylo možné provést numerickou simulaci.

Pro popis procesů v porézním prostředí z makroskopického hlediska je nutné zavést makroskopické veličiny, které popisují mikrostrukturu prostředí. Jednou z těchto veličin je *porozita*, která vyjadřuje podíl objemu pórů v části uvažované oblasti vzhledem k celkovému objemu této části. Uvažujme oblast $\Omega \subset \mathbb{R}^d$,



Obrázek 1.1: Závislost porozity Φ na průměru l referenční oblasti Ω_0.

kde $d = 1$, 2 nebo 3, vyplněnou porézním prostředím a definujme funkci

$$\gamma(\boldsymbol{x}) = \begin{cases} 1 & \text{pokud bod } \boldsymbol{x} \text{ leží v prázdném prostoru,} \\ 0 & \text{pokud bod } \boldsymbol{x} \text{ leží v prostoru vyplněném skeletem,} \end{cases} \qquad \forall \boldsymbol{x} \in \Omega, \tag{1.1}$$

přičemž rozhodování, zda bod $\boldsymbol{x}$ leží v prázdném prostoru nebo v prostoru vyplněném skeletem, provádíme na mikroskopické úrovni. Průměrováním funkce γ dospějeme k makroskopické veličině Φ zvané *porozita*. Uvažujme tedy podoblast $\Omega_0 \subset \Omega$ obsahující bod $\boldsymbol{x}_0$. Pak veličina Φ je dána vztahem

$$\Phi(\boldsymbol{x}_0) = \frac{1}{\int\limits_{\Omega_0} 1 \mathrm{d}\boldsymbol{x}} \int\limits_{\Omega_0} \gamma(\boldsymbol{x}) \mathrm{d}\boldsymbol{x}, \qquad \forall \boldsymbol{x}_0 \in \Omega. \tag{1.2}$$

Hodnota porozity Φ v daném bodě $\boldsymbol{x}_0$ závisí na velikosti oblasti Ω_0, přes kterou průměrujeme. Na obr. 1.1 je zobrazena závislost porozity Φ na průměru l oblasti Ω_0. Pro velmi malé hodnoty l vykazuje Φ výrazné oscilace způsobené nespojitostí funkce γ. Při překročení určité hodnoty l_{micro} se hodnota porozity ustálí na konstantní hodnotě a pro hodnoty $l > l_{macro}$ případné makroskopické nehomogenity opět destabilizují hodnotu Φ. Oblasti Ω_0, pro které existují hodnoty l_{micro} a l_{macro} takové, že pro $l_{micro} \ll l \ll l_{macro}$ hodnota Φ nezávisí na průměru oblasti l, označujeme jako *reprezentativní objem*, zkráceně REV[1]. Pokud pro uvažované porézní prostředí nelze pro nějaký bod $\boldsymbol{x}_0 \in \Omega$ určit reprezentativní objem, nelze ke studování proudění v tomto prostředí použít makroskopické metody.

1.2 Fáze

Fází rozumíme chemicky homogenní část zkoumaného systému, která se od ostatních částí systému liší svými fyzikálními vlastnostmi, a která je od ostatních částí systému oddělena jednoznačně určenou hranicí [2]. Podle definice jsou tedy fáze na mikroskopické úrovni nemísitelné. *Pevnou fází* rozumíme pevné látky, které tvoří skelet porézního prostředí. Obsahuje-li uvažovaný systém kromě pevné fáze pouze jednu další fázi, hovoříme o *jednofázovém systému*. Pokud systém obsahuje dvě nebo více dalších fází, jedná se o *vícefázový systém*.

Každou fázi lze uvažovat jako směs chemicky homogenních, na mikroskopické úrovni dokonale mísitelných částí, které nazýváme *komponentami* dané fáze. Příkladem fáze složené z více komponent může být směs plynů nebo voda obsahující rozpuštěný plyn. V takovém případě hovoříme o *kompozičním proudění*. Kromě pohybu komponent v rámci fáze můžou jednotlivé komponenty také přestupovat mezi fázemi, jako např. při rozpouštění vzduchu ve vodě nebo při odpařování vody.

[1] Representative Elementary Volume

1.3 Saturace

Abychom mohli určit rozložení fází v systému z makroskopického hlediska, definujeme vhodnou makroskopickou veličinu podobným způsobem, jak byla definována porozita. Zkoumejme reprezentativní objem $\Omega_0 \subset \Omega$ tvořený porézním materiálem vyplněným několika fázemi. V mikroskopickém měřítku je v každém bodě $\boldsymbol{x} \in \Omega_0$ přítomen buď skelet porézního prostředí, nebo právě jedna z fází. Lze tedy definovat indikátorovou funkci γ_α fáze α

$$\gamma_\alpha(\boldsymbol{x}, t) = \begin{cases} 1 & \text{pokud bod } \boldsymbol{x} \text{ leží v čase } t \text{ v prostoru vyplněném fází } \alpha, \\ 0 & \text{jinak,} \end{cases} \qquad \forall \boldsymbol{x} \in \Omega, \tag{1.3}$$

Pomocí funkce γ_α definujeme *saturaci* fáze α pomocí vztahu

$$S_\alpha(\boldsymbol{x}_0, t) = \frac{\int\limits_{\Omega_0} \gamma_\alpha(\boldsymbol{x}, t)}{\int\limits_{\Omega_0} \gamma(\boldsymbol{x}, t)}, \tag{1.4}$$

kde $\boldsymbol{x}_0 \in \Omega_0$. Veličina S_α je bezrozměrná makroskopická veličina vyjadřující poměr objemu obsazeného fází α ku objemu pórů v reprezentativním objemu Ω_0. Přímo z definice (1.4) plynou následující vlastnosti:

$$0 \leq S_\alpha \leq 1, \tag{1.5a}$$

$$\sum_\alpha S_\alpha = 1. \tag{1.5b}$$

1.3.1 Reziduální saturace

Je známo, že smáčivou fázi (vodu) nelze z porézního prostředí mechanicky úplně vytlačit. Tento fakt lze matematicky popsat zavedením tzv. *reziduální saturace* smáčivé fáze $S_{w,r}$, která vyjadřuje minimální množství fáze, které bude v porézním prostředí vždy přítomno. Pro popis opačných procesů lze definovat také reziduální saturaci nesmáčivé fáze $S_{n,r}$.

Pro práci s reziduálními saturacemi se dále zavádí *efektivní saturace* S^e_α fáze α,

$$S^e_\alpha = \frac{S_\alpha - S_{\alpha,r}}{1 - \sum\limits_\beta S_{\beta,r}}, \tag{1.6}$$

která postihuje pouze objemový podíl fáze, který lze mechanicky vytlačit. Přímo z definice plyne, že efektivní saturace má stejné vlastnosti jako saturace (1.5).

1.4 Zákon zachování hmoty

Zákony zachování hrají důležitou roli při popisu zachovávajících se veličin. Přirozeným požadavkem při popisu proudění několika nemísitelných fází je zachování hmoty každé z fází:

$$\frac{\partial(\phi S_\alpha \rho_\alpha)}{\partial t} + \nabla \cdot (\rho_\alpha \boldsymbol{v}_\alpha) = f_\alpha, \tag{1.7}$$

kde α je index dané fáze a veličiny vystupující v rovnici mají následující význam:

$\Phi(\boldsymbol{x})$	$[-]$	*Porozita* prostředí. Dále předpokládáme, že porozita nezávisí na čase, což odpovídá případu, kdy skelet porézního prostředí není deformován.
$S_\alpha(\boldsymbol{x}, t)$	$[-]$	*Saturace* fáze α.
$\rho_\alpha(\boldsymbol{x}, t)$	$[\mathrm{kg} \cdot \mathrm{m}^{-3}]$	*Hustota* fáze α.
$\boldsymbol{v}_\alpha(\boldsymbol{x}, t)$	$[\mathrm{m} \cdot \mathrm{s}^{-1}]$	*Průsaková rychlost*, neboli *Darcyho rychlost* fáze α.
$f_\alpha(\boldsymbol{x}, t)$	$[\mathrm{kg} \cdot \mathrm{m}^{-3} \cdot \mathrm{s}^{-1}]$	Člen charakterizující vnitřní zdroje a propady.

1.5 Darcyho zákon

Makroskopickou průsakovou rychlost $\boldsymbol{v}_\alpha$ každé fáze lze určit pomocí Darcyho zákona [11]:

$$\boldsymbol{v}_\alpha = -\frac{1}{\mu_\alpha}\mathbf{K}_\alpha(\nabla p_\alpha - \rho_\alpha \boldsymbol{g}), \tag{1.8}$$

kde nové veličiny mají následující význam:

$p_\alpha(\boldsymbol{x}, t)$	$[\mathrm{Pa}]$	*Tlak* fáze α.
μ_α	$[\mathrm{Pa} \cdot \mathrm{s}]$	*Dynamická viskozita* fáze α. Předpokládáme, že μ_α je konstantní.
$\mathbf{K}_\alpha(\boldsymbol{x})$	$[\mathrm{m}^2]$	*Propustnost* neboli *permeabilita* porézního prostředí vzhledem k fázi α, obecně se jedná o symetrický, pozitivně definitní tenzor 2. řádu.
$\boldsymbol{g}$	$[\mathrm{m} \cdot \mathrm{s}^{-2}]$	Vektor *gravitačního zrychlení*.

Darcyho zákon byl odvozen pro jednofázové proudění. Při rozšíření na vícefázové proudění se předpokládá závislost tenzoru $\mathbf{K}_\alpha$ nejen na vlastnostech prostředí, ale také na saturaci fáze α. Pokud jsou póry vyplněny více fázemi, nemůže fáze α využívat k proudění celý prostor pórů a výsledná průsaková rychlost $\boldsymbol{u}_\alpha$ je proto nižší, než když jsou póry vyplněny pouze jednou fází.

Při modelování se předpokládá, že tenzor $\mathbf{K}_\alpha$ lze rozložit jako součin $\mathbf{K}_\alpha = k_{r,\alpha}(S_\alpha)\mathbf{K}$, přičemž tenzor *absolutní permeability* (neboli vnitřní propustnosti) $\mathbf{K}$ $[\mathrm{m}^2]$ závisí pouze na vlastnostech prostředí a *relativní permeabilita* $k_{r,\alpha}$ $[-]$ na saturaci fáze α a implicitně také na vlastnostech prostředí. Relativní permeabilita představuje škálovací faktor splňující $0 \leq k_{r,\alpha}(S_\alpha) \leq 1$. Různé modely relativní permeability jsou popsány v sekci 1.7.

Darcyho zákon lze psát ve tvaru

$$\boldsymbol{v}_\alpha = -\frac{k_{r,\alpha}}{\mu_\alpha}\mathbf{K}(\nabla p_\alpha - \rho_\alpha \boldsymbol{g}). \tag{1.9}$$

Pro přehlednost se ještě zavádí značení $\lambda_\alpha = \frac{k_{r,\alpha}}{\mu_\alpha}$, kde λ_α $[\mathrm{Pa}^{-1} \cdot \mathrm{s}^{-1}]$ se nazývá *mobilita* fáze α.

1.6 Kapilární tlak

Ve zbytku této práce se omezíme na popis proudění dvou fází v porézním prostředí. Dvoufázové proudění v porézním prostředí je silně ovlivněno kapilaritou. Působení povrchového napětí na ostrém rozhraní mezi fázemi v pórech vyvolává rozdíl mezi mikroskopickými tlaky jednotlivých fází. Na fázovém

rozhraní definujeme *mikroskopický kapilární tlak* π_c jako rozdíl mezi tlakem nesmáčivé fáze π_n a tlakem smáčivé fáze π_w

$$\pi_c = \pi_n - \pi_w. \tag{1.10}$$

V důsledku definice smáčivosti je kapilární tlak vždy nezáporná veličina, $\pi_c \geq 0$. Dále bude vždy index n označovat nesmáčivou fázi (např. vzduch) a index w smáčivou fázi (např. voda).

Na makroskopické úrovni definujeme *makroskopický kapilární tlak* p_c přirozeně pomocí stejného vztahu jako v mikroskopickém případě

$$p_c = p_n - p_w, \tag{1.11}$$

kde makroskopické tlaky p_w a p_n lze definovat např. průměrováním příslušného mikroskopického tlaku přes reprezentativní objem (REV). Makroskopický kapilární tlak obecně závisí na saturacích fází, teplotě a rozměrech a uspořádání pórů. Dále uvažujeme pouze závislost kapilárního tlaku na saturacích, $p_c = p_c(S_w)$. Pro matematický popis této závislosti se nejčastěji používají modely van Genuchtena a Brookse a Coreyho.

Brooks a Corey

V modelu Brookse a Coreyho [6] je závislost kapilárního tlaku na saturaci S_w určena vztahem

$$p_c(S_w) = p_{B\&C}(S_w^e)^{-\frac{1}{\lambda_{B\&C}}}, \tag{1.12}$$

kde $p_{B\&C}\,[\mathrm{Pa}]$ a $\lambda_{B\&C}\,[-]$ jsou parametry modelu pro dané prostředí. Parametr $p_{B\&C}$ představuje vstupní tlak pro dané prostředí a $\lambda_{B\&C}$ souvisí s rozložením velikosti pórů.

Van Genuchten

Ve van Genuchtenově modelu [20] je závislost kapilárního tlaku na saturaci S_w určena vztahem

$$p_c(S_w) = \frac{1}{\alpha_{vG}}\left((S_w^e)^{-\frac{1}{m_{vG}}} - 1\right)^{\frac{1}{n_{vG}}}, \tag{1.13}$$

kde parametr $\alpha_{vG}\,[\mathrm{Pa}^{-1}]$ charakterizuje vstupní tlak a parametry $m_{vG}\,[-]$ a $n_{vG}\,[-]$ popisující strukturu pórů jsou obvykle svázány vztahem $m_{vG} = 1 - \frac{1}{n_{vG}}$.

1.7 Relativní permeabilita

Permeabilita prostředí vzhledem k fázi α byla zavedena v sekci 1.5 vztahem $\mathbf{K}_\alpha = k_{r,\alpha}(S_\alpha)\mathbf{K}$. Závislostí $k_{r,\alpha} = k_{r,\alpha}(S_\alpha)$ modelujeme skutečnost, že průchodu fáze α prostředím brání kromě skeletu také přítomnost ostatních fází. V případě dvoufázového proudění lze funkce $k_{r,w}$ a $k_{r,n}$ popisující závislost relativní permeability na saturaci odvodit z příslušného modelu kapilarity [23].

Burdine

Modelu Brookse a Coreyho odpovídá Burdineův model [7, 8]:

$$k_{r,w}(S_w) = (S_w^e)^{3+\frac{2}{\lambda_{B\&C}}}, \tag{1.14a}$$

$$k_{r,n}(S_n) = (S_n^e)^2\left(1 - (1 - S_n^e)^{1+\frac{2}{\lambda_{B\&C}}}\right), \tag{1.14b}$$

kde $\lambda_{B\&C}$ je parametr Brooksova a Coreyho modelu.

Mualem

Modelu van Genuchtena odpovídá Mualemův model [35]:

$$k_{r,w}(S_w) = (S_w^e)^{\frac{1}{2}} \left(1 - \left(1 - (S_w^e)^{\frac{1}{m_{vG}}}\right)^{m_{vG}}\right)^2, \tag{1.15a}$$

$$k_{r,n}(S_n) = (S_n^e)^{\frac{1}{3}} \left(1 - (1 - S_n^e)^{\frac{1}{m_{vG}}}\right)^{2m_{vG}}, \tag{1.15b}$$

kde m_{vG} je parametr van Genuchtenova modelu.

1.8 Rovnice dvoufázového proudění

Shrnutím předchozích sekcí dostáváme následující soustavu rovnic popisující nemísivé dvoufázové proudění v porézním prostředí:

$$\Phi \frac{\partial(S_w \rho_w)}{\partial t} + \nabla \cdot (\rho_w \boldsymbol{v}_w) = f_w, \tag{1.16a}$$

$$\boldsymbol{v}_w = -\lambda_w \mathbf{K}(\nabla p_w - \rho_w \boldsymbol{g}), \tag{1.16b}$$

$$\Phi \frac{\partial(S_n \rho_n)}{\partial t} + \nabla \cdot (\rho_n \boldsymbol{v}_n) = f_n, \tag{1.16c}$$

$$\boldsymbol{v}_n = -\lambda_n \mathbf{K}(\nabla p_n - \rho_n \boldsymbol{g}), \tag{1.16d}$$

$$p_c = p_n - p_w, \tag{1.16e}$$

$$S_w + S_n = 1, \tag{1.16f}$$

kde neznámé funkce jsou saturace $S_\alpha = S_\alpha(\boldsymbol{x}, t)$ a tlaky $p_\alpha = p_\alpha(\boldsymbol{x}, t)$ fází $\alpha \in \{w, n\}$ pro $\boldsymbol{x} \in \Omega$ a $t > 0$. Pro nestlačitelné fáze je soustava (1.16) kompletní, v případě stlačitelných fází by bylo potřeba předepsat ještě stavové rovnice určující závislost hustoty fáze na tlaku.

1.9 Transport komponent

Podobně jako při popisu proudění nemísitelných fází vyjdeme i při popisu proudění komponent ze zákona zachování hmoty. Zákon zachování hmoty komponenty κ ve fázi α uvažujeme ve tvaru advekčně–reakčně–difúzní rovnice [26]

$$\frac{\partial(\phi S_\alpha X_{\alpha,\kappa} \rho_\alpha)}{\partial t} + \nabla \cdot (\rho_\alpha \boldsymbol{v}_{\alpha,\kappa}) + \rho_\alpha r_{\alpha,\kappa} X_{\alpha,\kappa} = 0. \tag{1.17}$$

V případě dvoufázového kompozičního proudění přidáváme dle počtu komponent uvažovaných v systému příslušný počet rovnic typu (1.17) k rovnicím (1.16). Nové veličiny vystupující v rovnici (1.17) mají následující význam:

$X_{\alpha,\kappa}(\boldsymbol{x}, t)$	$[-]$	*Hmotnostní zlomek* komponenty κ ve fázi α.
$\boldsymbol{v}_{\alpha,\kappa}$	$[\mathrm{m} \cdot \mathrm{s}^{-1}]$	Rychlost komponenty κ ve fázi α.
$r_{\alpha,\kappa}$	$[\mathrm{s}^{-1}]$	Reakční člen popisující přestup komponent mezi fázemi.

Rychlost komponenty $\boldsymbol{v}_{\alpha,\kappa}$ budeme uvažovat ve tvaru

$$\boldsymbol{v}_{\alpha,\kappa} = X_{\alpha,\kappa}\boldsymbol{v}_\alpha - D_{\alpha,\kappa}\nabla X_{\alpha,\kappa}, \tag{1.18}$$

kde $\boldsymbol{v}_\alpha$ je rychlost fáze α daná Darcyho zákonem (1.8) a $D_{\alpha,\kappa}[\mathrm{m}^2 \cdot \mathrm{s}^{-1}]$ představuje koeficient difúze komponenty κ ve fázi α.

Pro modelování přestupu komponent mezi fázemi použijeme rovnovážný přístup [23]. Předpokládáme, že přestup komponent mezi fázemi probíhá mnohem rychleji oproti transportu komponent v jednotlivých fázích. Díky tomu jsou fáze neustále v lokální termodynamické rovnováze a přestup komponent mezi smáčivou a nesmáčivou fází můžeme popsat pomocí Henryho zákona [32]

$$X_{n,\kappa} = H_\kappa X_{w,\kappa}, \tag{1.19}$$

kde $H_\kappa[-]$ je bezrozměrná Henryho konstanta pro fázi κ.

Za předpokladu slabých roztoků můžeme hustoty ρ_w a ρ_n považovat za konstanty a eliminovat je z rovnice (1.17). Po dosazení vztahů (1.18) a (1.19) do (1.17) můžeme z výsledné rovnice eliminovat také Henryho konstantu H_κ. Dostáváme tak dvě rovnice pro $X_{w,\kappa}$, jejichž sečtením získáme

$$\phi\frac{\partial X}{\partial t} + \nabla \cdot (X\boldsymbol{v}_t - D_X\nabla X) + r_X X = 0, \tag{1.20}$$

kde jsme použili označení $\boldsymbol{v}_t = \boldsymbol{v}_w + \boldsymbol{v}_n$, $X = X_{w,\kappa}$, $D_X = D_{w,\kappa} + D_{n,\kappa}$ a $r_X = r_{w,\kappa} + r_{n,\kappa}$.

Kapitola 2

Obecná formulace

Pro řešení úloh dvoufázového a kompozičního proudění v porézním prostředí přejdeme k jednotné formulaci publikované v článku [16]. Uvažujeme obecnou soustavu n parciálních diferenciálních rovnic ve tvaru

$$\sum_{j=1}^{n} N_{i,j}\frac{\partial Z_j}{\partial t} + \sum_{j=1}^{n} \boldsymbol{u}_{i,j}\cdot\nabla Z_j + \nabla\cdot\left[m_i\left(-\sum_{j=1}^{n}\mathbf{D}_{i,j}\nabla Z_j + \boldsymbol{w}_i\right) + \sum_{j=1}^{n} Z_j\boldsymbol{a}_{i,j}\right] + \sum_{j=1}^{n} r_{i,j}Z_j = f_i \tag{2.1}$$

pro $i = 1, ..., n$, kde $\boldsymbol{Z} = (Z_1, ..., Z_n)^T$ je vektor neznámých funkcí závislých na poloze $\boldsymbol{x} \in \Omega \subset \mathbb{R}^d$ a čase $t \in (0, T)$, kde $d = 1$, 2 nebo 3 označuje dimenzi prostoru a $T > 0$ určuje konečný čas. Soustava (2.1) je doplněna o počáteční podmínku

$$Z_j(\boldsymbol{x}, 0) = Z_j^{ini}(\boldsymbol{x}), \quad \forall \boldsymbol{x} \in \Omega,\ j = 1, \ldots, n, \tag{2.2}$$

a okrajové podmínky pro všechna $t \in (0, T)$,

$$Z_j(\boldsymbol{x}, t) = Z_j^{\mathcal{D}}(\boldsymbol{x}, t), \quad \forall \boldsymbol{x} \in \Gamma_j^{\mathcal{D}} \subset \partial\Omega,\ \ j = 1, ..., n, \tag{2.3a}$$

$$\boldsymbol{v}_i(\boldsymbol{x}, t)\cdot\boldsymbol{n}_{\partial\Omega}(\boldsymbol{x}) = v_i^{\mathcal{N}}(\boldsymbol{x}, t), \quad \forall \boldsymbol{x} \in \Gamma_i^{\mathcal{N}} \subset \partial\Omega,\ \ i = 1, ..., n, \tag{2.3b}$$

kde $\boldsymbol{v}_i$ označuje rychlostní člen

$$\boldsymbol{v}_i = -\sum_{j=1}^{n}\mathbf{D}_{i,j}\nabla Z_j + \boldsymbol{w}_i, \tag{2.4}$$

a rozklad hranice $\partial\Omega$ pro všechna $j = 1, \ldots, n$ splňuje

$$\Gamma_j^{\mathcal{D}} \cup \Gamma_j^{\mathcal{N}} = \partial\Omega, \tag{2.5a}$$

$$\Gamma_j^{\mathcal{D}} \cap \Gamma_j^{\mathcal{N}} = \emptyset. \tag{2.5b}$$

Koeficienty vystupující v soustavě (2.1) mohou obecně záviset na vektoru neznámých funkcí $\boldsymbol{Z}$ a mají následující význam: $\mathbf{N} = [N_{i,j}]_{i,j=1}^n$ představuje matici útlumu, $\boldsymbol{u} = [\boldsymbol{u}_{i,j}]_{i,j=1}^n$ popisuje konvekci v nekonzervativní formě, $\boldsymbol{m} = [m_i]_{i=1}^n$ je nezáporný vektor koeficientů mobility, $\mathbf{D} = [\mathbf{D}_{i,j}]_{i,j=1}^n$ je matice nulových nebo symetrických a pozitivně definitních tenzorů difúze, přičemž předpokládáme, že pro každé i existuje alespoň jeden index j takový, že $\mathbf{D}_{i,j}$ je nenulový, $\boldsymbol{w} = [\boldsymbol{w}_i]_{i=1}^n$ představuje vektor externích konzervativních sil, $\boldsymbol{a} = [\boldsymbol{a}_{i,j}]_{i,j=1}^n$ popisuje konvekci v konzervativní formě, $\boldsymbol{r} = [r_{i,j}]_{i,j=1}^n$ je matice reakčních členů a $\boldsymbol{f} = [f_i]_{i=1}^n$ představuje vektor zdrojových členů.

Volba koeficientů a neznámých v obecné rovnici (2.1) závisí na konkrétní úloze a její formulaci. V následujících sekcích uvedeme volbu koeficientů pro úlohy nemísivého dvoufázového proudění a kompozičního dvoufázového proudění v porézním prostředí, kterými se budeme zabývat v dalších kapitolách.

2.1 Dvoufázové proudění

Při volbě koeficientů rovnice (2.1) pro úlohu nestlačitelného a nemísivého dvoufázového proudění v porézním prostředí bez vnitřních zdrojů a propadů vyjdeme z rovnice (1.16). Máme $n = 2$ a s ohledem na použité numerické schéma zvolíme za primární neznámé funkce fázové tlaky, $Z_1 = p_w$ a $Z_2 = p_n$. Po vyjádření zbylých neznámých pomocí fázových tlaků a provedení úprav získáme následující volbu koeficientů:

$$\mathbf{N} = \begin{pmatrix} -\Phi\frac{\mathrm{d}S_w}{\mathrm{d}p_c} & \Phi\frac{\mathrm{d}S_w}{\mathrm{d}p_c} \\ \Phi\frac{\mathrm{d}S_w}{\mathrm{d}p_c} & -\Phi\frac{\mathrm{d}S_w}{\mathrm{d}p_c} \end{pmatrix}, \tag{2.6a}$$

$$\boldsymbol{u} = \begin{pmatrix} \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{pmatrix}, \tag{2.6b}$$

$$\boldsymbol{m} = \begin{pmatrix} \frac{\lambda_w}{\lambda_t} \\ \frac{\lambda_n}{\lambda_t} \end{pmatrix}, \tag{2.6c}$$

$$\mathbf{D} = \begin{pmatrix} \lambda_t\mathbf{K} & \mathbf{0} \\ \mathbf{0} & \lambda_t\mathbf{K} \end{pmatrix}, \tag{2.6d}$$

$$\boldsymbol{w} = \begin{pmatrix} \lambda_t\rho_w\mathbf{K}\boldsymbol{g} \\ \lambda_t\rho_n\mathbf{K}\boldsymbol{g} \end{pmatrix}, \tag{2.6e}$$

$$\boldsymbol{a} = \begin{pmatrix} \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{pmatrix}, \tag{2.6f}$$

$$\boldsymbol{r} = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}, \tag{2.6g}$$

$$\boldsymbol{f} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \tag{2.6h}$$

kde $\lambda_t = \lambda_w + \lambda_n$. Pro volbu koeficientů $\boldsymbol{m}$ a $\mathbf{D}$ byl využit postup z článku [25].

2.2 Kompoziční proudění

Pro úlohu kompozičního dvoufázového proudění v porézním prostředí využijeme koeficienty uvedené v předchozí sekci pro popis dvoufázového proudění, ke kterým přidáme koeficienty příslušející transportní rovnici (1.20).

Transportní rovnici (1.20) můžeme zapsat ve tvaru rovnice (2.1) buď v konzervativním nebo v nekonzervativním tvaru. Konzervativní tvar je dán přímo rovnicí (1.20). Sečtením zákonů zachování hmoty (1.7) pro obě fáze s využitím identity $\partial(S_w + S_n)/\partial t = 0$ dostaneme, že $\nabla \cdot \boldsymbol{v}_t = 0$. Použitím této identity v rovnici (1.20) dostaneme její nekonzervativní tvar:

$$\phi\frac{\partial X}{\partial t} + \boldsymbol{v}_t \cdot \nabla X - \nabla \cdot (D_X \nabla X) + r_X X = 0. \tag{2.7}$$

Transportní rovnice (1.20) umožňuje použít degenerující nebo nulový difúzní člen D_X a řešit tak kromě difúzních úloh také úlohy s dominantní advekcí nebo i čistě advekční úlohy. Abychom mohli simulovat degenerující nebo nulovou difúzi pomocí schématu prezentovaného v kapitole 3, uvažujeme rozklad $D_X = m_X D_0$, kde $D_0 > 0$ je konstantní difúzní koeficient a $m_X \geq 0$ je koeficient mobility. Volbou konstanty D_0 regularizujeme numerické schéma a mobilitu m_X můžeme volit libovolně malou pro případ degenerující difúze, nebo i nulovou v případě čistě advekční úlohy.

Pro konzervativní i nekonzervativní variantu máme $n = 3$, primární proměnné $Z_1 = p_w$, $Z_2 = p_n$, $Z_3 = X$ a příslušné koeficienty v rovnici (2.1) jsou:

Konzervativní forma:

$$\mathbf{N} = \begin{pmatrix} -\Phi\frac{\mathrm{d}S_w}{\mathrm{d}p_c} & \Phi\frac{\mathrm{d}S_w}{\mathrm{d}p_c} & 0 \\ \Phi\frac{\mathrm{d}S_w}{\mathrm{d}p_c} & -\Phi\frac{\mathrm{d}S_w}{\mathrm{d}p_c} & 0 \\ 0 & 0 & \Phi \end{pmatrix}, \tag{2.8a}$$

$$\boldsymbol{u} = \begin{pmatrix} \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \end{pmatrix}, \tag{2.8b}$$

$$\boldsymbol{m} = \left(\frac{\lambda_w}{\lambda_t},\ \frac{\lambda_n}{\lambda_t},\ m_X\right)^T, \tag{2.8c}$$

$$\mathbf{D} = \begin{pmatrix} \lambda_t\mathbf{K} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \lambda_t\mathbf{K} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & D_0\mathbf{I} \end{pmatrix}, \tag{2.8d}$$

$$\boldsymbol{w} = (-\lambda_t\rho_w\mathbf{K}\boldsymbol{g},\ \lambda_t\rho_n\mathbf{K}\boldsymbol{g},\ \mathbf{0})^T, \tag{2.8e}$$

$$\boldsymbol{a} = \begin{pmatrix} \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \boldsymbol{v}_t \end{pmatrix}, \tag{2.8f}$$

$$\boldsymbol{r} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & r_X \end{pmatrix}, \tag{2.8g}$$

$$\boldsymbol{f} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, \tag{2.8h}$$

Nekonzervativní forma:

$$\mathbf{N} = \begin{pmatrix} -\Phi\frac{\mathrm{d}S_w}{\mathrm{d}p_c} & \Phi\frac{\mathrm{d}S_w}{\mathrm{d}p_c} & 0 \\ \Phi\frac{\mathrm{d}S_w}{\mathrm{d}p_c} & -\Phi\frac{\mathrm{d}S_w}{\mathrm{d}p_c} & 0 \\ 0 & 0 & \Phi \end{pmatrix}, \tag{2.8a}$$

$$\boldsymbol{u} = \begin{pmatrix} \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \boldsymbol{v}_t \end{pmatrix}, \tag{2.8b}$$

$$\boldsymbol{m} = \left(\frac{\lambda_w}{\lambda_t},\ \frac{\lambda_n}{\lambda_t},\ m_X\right)^T, \tag{2.8c}$$

$$\mathbf{D} = \begin{pmatrix} \lambda_t\mathbf{K} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \lambda_t\mathbf{K} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & D_0\mathbf{I} \end{pmatrix}, \tag{2.8d}$$

$$\boldsymbol{w} = (-\lambda_t\rho_w\mathbf{K}\boldsymbol{g},\ \lambda_t\rho_n\mathbf{K}\boldsymbol{g},\ \mathbf{0})^T, \tag{2.8e}$$

$$\boldsymbol{a} = \begin{pmatrix} \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \end{pmatrix}, \tag{2.8f}$$

$$\boldsymbol{r} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & r_X \end{pmatrix}, \tag{2.8g}$$

$$\boldsymbol{f} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}. \tag{2.8h}$$

Kapitola 3

Numerické schéma

V této kapitole popíšeme numerické schéma pro řešení soustavy (2.1). Schéma je založeno na kombinaci hybridní metody smíšených konečných prvků (MHFEM[1]) a nespojité Galerkinovy metody (DGM[2]) pro prostorovou diskretizaci, Eulerovy metody pro časovou diskretizaci a semi-implicitním přístupu metody zamrzlých koeficientů pro linearizaci v čase. Toto obecné schéma jsme publikovali v článku [16], podobnou formulaci numerického schématu pro nemísivé dvoufázové proudění lze nalézt v [17, 18, 25].

Uvažujeme diskretizaci prostorové oblasti $\Omega \subset \mathbb{R}^d$ pomocí numerické sítě $\mathcal{K}_h$ tvořené konečným počtem *elementů* K, kde $h > 0$ označuje průměr největšího elementu. Předpokládáme, že síť je konformní, tj. průnik uzávěrů dvou různých elementů je buď prázdný, vrchol, hrana nebo stěna. V aplikacích numerického schématu popsaných v dalších kapitolách této práce používáme buď strukturované nebo nestrukturované sítě tvořené vždy stejným typem elementů, a to buď obdélníky nebo trojúhelníky v $\mathbb{R}^2$ a kvádry nebo čtyřstěny v $\mathbb{R}^3$.

Nadstěnou[3] elementu K rozumíme geometrický útvar dimenze $d-1$, který tvoří část hranice ∂K. V následujících sekcích značíme množinu nadstěn elementu $K \in \mathcal{K}_h$ symbolem $\mathcal{E}_K$. Pro množinu nadstěn všech elementů numerické sítě $\mathcal{K}_h$ zavedeme symbol $\mathcal{E}_h$. Každá nadstěna $E \in \mathcal{E}_h$ je buď vnitřní nebo okrajová, množinu vnitřních nadstěn značíme symbolem $\mathcal{E}_h^{int}$ a množinu vnějších nadstěn značíme symbolem $\mathcal{E}_h^{ext}$. Počet prvků diskrétní množiny $\mathcal{A}$ značíme symbolem $\#\mathcal{A}$.

3.1 Variační formulace

Předpokládáme, že skalární funkce $N_{i,j}$, m_i, $r_{i,j}$, f_i a Z_j jsou pro všechna $i,j = 1,\dots,n$ prvky funkčního prostoru $L^2(\Omega)$. Dále předpokládáme, že vektorové funkce $\boldsymbol{u}_{i,j}$, $\boldsymbol{a}_{i,j}$ a $\boldsymbol{v}_i$ jsou pro všechna $i,j = 1,\dots,n$ prvky funkčního prostoru $\mathbf{H}(\mathrm{div},\Omega)$, kde

$$\mathbf{H}(\mathrm{div},\Omega) = \left\{ \boldsymbol{\omega} \in [L^2(\Omega)]^d \mid \nabla\cdot\boldsymbol{\omega} \in L^2(\Omega) \right\}. \tag{3.1}$$

Smíšenou variační formulaci úlohy (2.1) a (2.4) lze formálně odvodit vynásobením rovnice (2.1) funkcí $\varphi \in V \equiv L^2(\Omega)$ a rovnice (2.4) funkcí $\boldsymbol{\omega} \in \mathbf{W} \equiv \mathbf{H}(\mathrm{div},\Omega)$ a integrací obou rovnic přes oblast Ω. Pro nalezení přibližného řešení $Z_i \in V$, $\boldsymbol{v}_i \in \mathbf{W}$, $i = 1,\dots,n$ variační úlohy pomocí metody smíšených konečných prvků přejdeme k vhodným konečněrozměrným podprostorům $V_h \subset V$, $\mathbf{W}_h \subset \mathbf{W}$. V této práci použijeme prostory nejnižšího řádu, prostor po částech konstantních funkcí $D_0(\mathcal{K}_h)$ a Raviartův–Thomasův–Nédélecův prostor $\mathbf{RTN}_0(\mathcal{K}_h)$.

[1] Mixed Hybrid Finite Element Method

[2] Discontinuous Galerkin Method

[3] Anglicky *facet*

3.1.1 Prostor RTN_0

Při definici Raviartova–Thomasova–Nédélecova prostoru $\mathbf{RTN}_0(\mathcal{K}_h)$ využijeme diskretizaci oblasti Ω pomocí numerické sítě $\mathcal{K}_h$. Nejprve na každém elementu $K \in \mathcal{K}_h$ definujeme lokální prostor $\mathbf{RTN}_0(K)$ generovaný bazickými funkcemi $\boldsymbol{\omega}_{K,E} \in \mathbf{H}(\mathrm{div}, K)$, $E \in \mathcal{E}_K$. Konkrétní podoba prostoru $\mathbf{RTN}_0(K)$ závisí na dimenzi prostoru a na geometrii elementu K, obecně je však výhodné volit bazické funkce $\boldsymbol{\omega}_{K,E}$ tak, aby splňovaly podmínky

$$\nabla \cdot \boldsymbol{\omega}_{K,E}(\boldsymbol{x}) = \frac{1}{|K|_d}, \qquad \forall E \in \mathcal{E}_K, \forall \boldsymbol{x} \in K, \tag{3.2a}$$

$$\boldsymbol{\omega}_{K,E}(\boldsymbol{x}) \cdot \boldsymbol{n}_{K,F} = \frac{1}{|F|_{d-1}} \delta_{EF}, \qquad \forall E, F \in \mathcal{E}_K, \forall \boldsymbol{x} \in F, \tag{3.2b}$$

kde $\boldsymbol{n}_{K,F}$ označuje jednotkový vektor vnější normály na nadstěně $F \in \mathcal{E}_K$ vzhledem k elementu K, δ_{EF} představuje Kroneckerovo delta a $|\cdot|_s$ značí s-rozměrnou Lebesgueovu míru, $s = 0, 1, 2, 3$ a $|\cdot|_0 \equiv 1$.

Pomocí lokálních prostorů $\mathbf{RTN}_0(K)$ definujeme prostor $\mathbf{RTN}_0(\mathcal{K}_h)$ tak, aby se jednalo o podprostor prostoru $\mathbf{H}(\mathrm{div}, \Omega)$. Podle definice platí $\mathbf{RTN}_0(K) \subset \mathbf{H}(\mathrm{div}, K)$. Definujme tedy prostor

$$\mathbf{RTN}_0^0(\mathcal{K}_h) = \left\{ \boldsymbol{\omega} \in [L^2(\Omega)]^d \;\middle|\; \forall K \in \mathcal{K}_h,\ \boldsymbol{\omega}|_K \in \mathbf{RTN}_0(K) \right\}. \tag{3.3}$$

Prostor $\mathbf{RTN}_0^0(\mathcal{K}_h)$ však není podprostorem prostoru $\mathbf{H}(\mathrm{div}, \Omega)$. Ve smyslu tvrzení 1.2 v knize [5, str. 95] navíc po funkcích $\boldsymbol{\omega} \in \mathbf{RTN}_0^0(\mathcal{K}_h)$ požadujeme spojitost normálových stop na vnitřních nadstěnách sítě. Tuto vlastnost můžeme ekvivalentně vyjádřit pomocí bilanční podmínky, kterou použijeme pro definici prostoru $\mathbf{RTN}_0(\mathcal{K}_h)$:

$$\mathbf{RTN}_0(\mathcal{K}_h) = \left\{ \boldsymbol{\omega} \in [L^2(\Omega)]^d \;\middle|\; \forall K \in \mathcal{K}_h,\ \boldsymbol{\omega}|_K \in \mathbf{RTN}_0(K);\ \forall E \in \mathcal{E}_h^{int}, E \in \mathcal{E}_{K_1} \cap \mathcal{E}_{K_2}, \right.$$
$$\left. \int_E \boldsymbol{\omega}|_{K_1} \cdot \boldsymbol{n}_{K_1,E} + \int_E \boldsymbol{\omega}|_{K_2} \cdot \boldsymbol{n}_{K_2,E} = 0 \right\}. \tag{3.4}$$

Bazické funkce pro simplexy v $\mathbb{R}^d$

Simplex $K \in \mathcal{K}_h$ v $\mathbb{R}^d$ je konvexní obal $d+1$ afinně nezávislých bodů $\boldsymbol{V}_0, \ldots, \boldsymbol{V}_d \in \mathbb{R}^d$. Předpokládáme, že nadstěny simplexu $E_0, \ldots, E_d \in \mathcal{E}_K$ jsou číslovány tak, že pro všechna $\iota = 0, \ldots, d$ je $\boldsymbol{V}_\iota$ protější vrchol nadstěny E_ι. Potom bazické funkce prostoru $\mathbf{RTN}_0(K)$ splňující vlastnosti (3.2) mají tvar

$$\boldsymbol{\omega}_{K,E_\iota}(\boldsymbol{x}) = \frac{1}{d|K|_d} (\boldsymbol{x} - \boldsymbol{V}_\iota), \tag{3.5}$$

pro všechna $\iota = 0, \ldots, d$ a $x \in K$.

Bazické funkce pro obdélníky

Pro referenční obdélníkový element $K = (0, h_x) \times (0, h_y)$ v $\mathbb{R}^2$ volíme bazické funkce prostoru $\mathbf{RTN}_0(K)$ ve tvaru

$$\begin{aligned} \boldsymbol{\omega}_{K,E_1}(\boldsymbol{x}) &= \frac{1}{|K|_2}\begin{pmatrix} x_1 - h_x \\ 0 \end{pmatrix}, & \boldsymbol{\omega}_{K,E_2}(\boldsymbol{x}) &= \frac{1}{|K|_2}\begin{pmatrix} x_1 \\ 0 \end{pmatrix}, \\ \boldsymbol{\omega}_{K,E_3}(\boldsymbol{x}) &= \frac{1}{|K|_2}\begin{pmatrix} 0 \\ x_2 - h_y \end{pmatrix}, & \boldsymbol{\omega}_{K,E_4}(\boldsymbol{x}) &= \frac{1}{|K|_2}\begin{pmatrix} 0 \\ x_2 \end{pmatrix}, \end{aligned} \tag{3.6}$$

kde x_1, x_2 značí složky vektoru $\boldsymbol{x} \in K \subset \mathbb{R}^2$ a E_1, E_2, E_3, E_4 značí postupně levou, pravou, dolní a horní hranu elementu K.

Bazické funkce pro kvádry

Pro referenční kvádrový element $K = (0, h_x) \times (0, h_y) \times (0, h_z)$ v $\mathbb{R}^3$ volíme bazické funkce prostoru $\mathbf{RTN}_0(K)$ ve tvaru

$$\begin{aligned} \boldsymbol{\omega}_{K,E_1}(\boldsymbol{x}) &= \frac{1}{|K|_3}\begin{pmatrix} x_1 - h_x \\ 0 \\ 0 \end{pmatrix}, & \boldsymbol{\omega}_{K,E_2}(\boldsymbol{x}) &= \frac{1}{|K|_3}\begin{pmatrix} x_1 \\ 0 \\ 0 \end{pmatrix}, \\ \boldsymbol{\omega}_{K,E_3}(\boldsymbol{x}) &= \frac{1}{|K|_3}\begin{pmatrix} 0 \\ x_2 - h_y \\ 0 \end{pmatrix}, & \boldsymbol{\omega}_{K,E_4}(\boldsymbol{x}) &= \frac{1}{|K|_3}\begin{pmatrix} 0 \\ x_2 \\ 0 \end{pmatrix}, \\ \boldsymbol{\omega}_{K,E_5}(\boldsymbol{x}) &= \frac{1}{|K|_3}\begin{pmatrix} 0 \\ 0 \\ x_3 - h_z \end{pmatrix}, & \boldsymbol{\omega}_{K,E_6}(\boldsymbol{x}) &= \frac{1}{|K|_3}\begin{pmatrix} 0 \\ 0 \\ x_3 \end{pmatrix}, \end{aligned} \tag{3.7}$$

kde x_1, x_2, x_3 značí složky vektoru $\boldsymbol{x} \in K \subset \mathbb{R}^3$ a E_ι jsou stěny elementu K číslované následujícím způsobem:

- E_1 a E_2 jsou stěny kolmé na osu x_1, stěna E_1 prochází bodem $\boldsymbol{x} = \mathbf{0}$,
- E_3 a E_4 jsou stěny kolmé na osu x_2, stěna E_3 prochází bodem $\boldsymbol{x} = \mathbf{0}$,
- E_5 a E_6 jsou stěny kolmé na osu x_3, stěna E_5 prochází bodem $\boldsymbol{x} = \mathbf{0}$.

3.1.2 Aproximace vektorových funkcí

Vektorové funkce $\boldsymbol{v}_i$, $\boldsymbol{u}_{i,j}$ a $\boldsymbol{a}_{i,j}$ aproximujeme v prostoru $\mathbf{RTN}_0(\mathcal{K}_h)$. Vyjádřením v bázi prostoru $\mathbf{RTN}_0^0(\mathcal{K}_h)$ získáme

$$\boldsymbol{v}_i(\boldsymbol{x}, t) \approx \sum_{K \in \mathcal{K}_h} \sum_{E \in \mathcal{E}_K} v_{i,K,E}(t) \boldsymbol{\omega}_{K,E}(\boldsymbol{x}), \tag{3.8a}$$

$$\boldsymbol{a}_{i,j}(\boldsymbol{x}, t) \approx \sum_{K \in \mathcal{K}_h} \sum_{E \in \mathcal{E}_K} a_{i,j,K,E}(t) \boldsymbol{\omega}_{K,E}(\boldsymbol{x}), \tag{3.8b}$$

$$\boldsymbol{u}_{i,j}(\boldsymbol{x}, t) \approx \sum_{K \in \mathcal{K}_h} \sum_{E \in \mathcal{E}_K} u_{i,j,K,E}(t) \boldsymbol{\omega}_{K,E}(\boldsymbol{x}), \tag{3.8c}$$

$$\boldsymbol{w}_i(\boldsymbol{x}, t) \approx \sum_{K \in \mathcal{K}_h} \sum_{E \in \mathcal{E}_K} w_{i,K,E}(t) \boldsymbol{\omega}_{K,E}(\boldsymbol{x}). \tag{3.8d}$$

kde $\omega_{K,E}$ jsou bazické funkce prostoru $\mathbf{RTN}_0(K)$. Integrací vnější normálové stopy $\boldsymbol{v}_i|_K \cdot \boldsymbol{n}_{K,E}$ vzhledem k elementu K přes nadstěnu $E \in \mathcal{E}_K$ a s využitím vlastností bazických funkcí (3.2) dostaneme

$$\int_E \boldsymbol{v}_i|_K \cdot \boldsymbol{n}_{K,E} \approx \int_E \sum_{F \in \mathcal{E}_K} v_{i,K,F} \boldsymbol{\omega}_{K,F} \cdot \boldsymbol{n}_{K,E} = v_{i,K,E}. \tag{3.9}$$

Analogické vztahy platí i pro funkce $\boldsymbol{a}_{i,j}$, $\boldsymbol{u}_{i,j}$ a $\boldsymbol{w}_i$. Koeficienty $v_{i,K,E}$, $u_{i,j,K,E}$ a $a_{i,j,K,E}$ tedy odpovídají numerickému toku příslušné funkce přes nadstěnu $E \in \mathcal{E}_K$ ve směru vnější normály vzhledem k elementu K. Abychom získali aproximaci funkcí v prostoru $\mathbf{RTN}_0(\mathcal{K}_h)$, musí splňovat podmínku spojitosti normálových stop na vnitřních nadstěnách sítě. Na nadstěně E oddělující elementy K_1 a K_2 tedy musí platit

$$v_{i,K_1,E} + v_{i,K_2,E} = 0, \tag{3.10a}$$
$$a_{i,j,K_1,E} + a_{i,j,K_2,E} = 0, \tag{3.10b}$$
$$u_{i,j,K_1,E} + u_{i,j,K_2,E} = 0, \tag{3.10c}$$
$$w_{i,K_1,E} + w_{i,K_2,E} = 0. \tag{3.10d}$$

Tyto podmínky odpovídají předpokladu nulových zdrojových členů na nadstěně E a představují tedy lokální bilanci veličin $\boldsymbol{v}_i$, $\boldsymbol{a}_{i,j}$, $\boldsymbol{u}_{i,j}$ a $\boldsymbol{w}_i$.

3.1.3 Aproximace skalárních funkcí

Pro aproximaci skalárních funkcí uvažujeme konečněrozměrný prostor $V_h = D_0(\mathcal{K}_h)$ funkcí po částech konstantních na elementech $K \in \mathcal{K}_h$, jehož bazické funkce jsou

$$\varphi_K(\boldsymbol{x}) = \begin{cases} 1 & \text{pokud } \boldsymbol{x} \in K, \\ 0 & \text{jinak.} \end{cases} \tag{3.11}$$

Důležité vlastnosti bazických funkcí prostoru $D_0(\mathcal{K}_h)$ jsou:

$$\operatorname{supp} \varphi_K \subseteq K, \quad \int_\Omega \varphi_K = |K|_d, \quad \nabla \varphi_K = \mathbf{0}. \tag{3.12}$$

Neznámé Z_j a skalární koeficienty $N_{i,j}$, m_i, $r_{i,j}$ a f_i rovnice (2.1) aproximujeme pomocí projekce do prostoru $D_0(\mathcal{K}_h)$, tedy pro všechna $\boldsymbol{x} \in \Omega$ a $t \in (0,T)$

$$Z_j(\boldsymbol{x},t) \approx \sum_{K \in \mathcal{K}_h} Z_{j,K}(t) \varphi_K(\boldsymbol{x}), \tag{3.13a}$$
$$N_{i,j}(\boldsymbol{x},t) \approx \sum_{K \in \mathcal{K}_h} N_{i,j,K}(t) \varphi_K(\boldsymbol{x}), \tag{3.13b}$$
$$m_i(\boldsymbol{x},t) \approx \sum_{K \in \mathcal{K}_h} m_{i,K}(t) \varphi_K(\boldsymbol{x}), \tag{3.13c}$$
$$r_{i,j}(\boldsymbol{x},t) \approx \sum_{K \in \mathcal{K}_h} r_{i,j,K}(t) \varphi_K(\boldsymbol{x}), \tag{3.13d}$$
$$f_i(\boldsymbol{x},t) \approx \sum_{K \in \mathcal{K}_h} f_{i,K}(t) \varphi_K(\boldsymbol{x}). \tag{3.13e}$$

Koeficienty $Z_{j,K}$, $N_{i,j,K}$, $m_{i,K}$, $r_{i,j,K}$ a $f_{i,K}$ vystupující v (3.13) závisí na čase a lze je interpretovat jako střední hodnoty příslušné veličiny na elementu $K \in \mathcal{K}_h$. Variační aproximace v prostoru po částech konstantních funkcí tedy odpovídá použití metody konečných objemů s centrem konečného objemu v těžišti elementu (CFVM[4]).

3.1.4 Diskretizace rovnice pro rychlostní člen

Rovnici (2.4) můžeme formálně diskretizovat vynásobením bazickou funkcí prostoru $\boldsymbol{\omega} \in \mathbf{W}_h$ a integrací přes oblast Ω, čímž získáme soustavu $n \times \#\mathcal{E}_h$ rovnic pro $n \times (\#\mathcal{E}_h + \#\mathcal{K}_h)$ neznámých $v_{i,K,E}$ a $Z_{j,K}$, kde $i,j = 1,\dots,n$, $E \in \mathcal{E}_h$ a $K \in \mathcal{K}_h$. Tato přímočará aplikace metody smíšených konečných prvků však vede na soustavu s indefinitní maticí [5]. Hybridní metoda smíšených konečných prvků dále využívá blokové struktury této soustavy a pro koeficienty $v_{i,K,E}$ najde explicitní závislost na diskretizovaných neznámých Z_j. Pro formulaci numerického schématu pro řešení soustavy (2.1) použijeme metodu hybridizace dále popsaným způsobem.

Definujme parciální rychlosti $\boldsymbol{v}_{i,j}$ a předpokládejme, že $\boldsymbol{v}_{i,j}$ patří do prostoru $\mathbf{H}(\mathrm{div},\Omega)$:

$$\boldsymbol{v}_{i,j} = -\mathbf{D}_{i,j}\nabla Z_j. \tag{3.14}$$

Aproximaci $\boldsymbol{v}_{i,j}$ v lokálním prostoru $\mathbf{RTN}_0(K)$ můžeme zapsat pomocí

$$\boldsymbol{v}_{i,j}|_K \approx \sum_{E\in\mathcal{E}_K} v_{i,j,K,E}\boldsymbol{\omega}_{K,E}, \tag{3.15}$$

kde $\boldsymbol{\omega}_{K,E}$ označuje bazickou funkci prostoru $\mathbf{RTN}_0(K)$. Předpokládáme, že tenzor $\mathbf{D}_{i,j}$ je na elementu K buď nulový, nebo symetrický a pozitivně definitní. Pokud je $\mathbf{D}_{i,j}|_K$ nulový, je i projekce $\boldsymbol{v}_{i,j}|_K$ nulová, a tedy i koeficienty $v_{i,j,K,E}$ musí být nulové. Pokud je $\mathbf{D}_{i,j}|_K$ symetrický a pozitivně definitní, můžeme rovnici (3.14) vynásobit jeho inverzí a požadovat splnění výsledné rovnosti ve slabém smyslu:

$$\sum_{E\in\mathcal{E}_K} v_{i,j,K,E}\int_K \boldsymbol{\omega}_{K,F}^T\mathbf{D}_{i,j}^{-1}\boldsymbol{\omega}_{K,E} = -\int_K \boldsymbol{\omega}_{K,F}^T\nabla Z_j, \qquad \forall F\in\mathcal{E}_K, \tag{3.16}$$

kde $\boldsymbol{\omega}_{K,F}$ označuje bazickou funkci prostoru $\mathbf{RTN}_0(K)$. Dále pro úpravu pravé strany (3.16) použijeme Greenovu formuli, vztah (3.13a), vlastnosti bazických funkcí (3.2) a (3.12):

$$-\int_K \boldsymbol{\omega}_{K,F}^T\nabla Z_j = -\int_{\partial K} Z_j\boldsymbol{\omega}_{K,F}^T\boldsymbol{n}_{\partial K} + \int_K Z_j\nabla\cdot\boldsymbol{\omega}_{K,F} = -Z_{j,F} + Z_{j,K}, \tag{3.17}$$

kde koeficienty $Z_{j,F}$ přestavují nové stupně volnosti veličiny Z_j na nadstěnách sítě. Dle [5, kap. 5.1.2] mají koeficienty $Z_{j,F}$ ve schématu roli multiplikátorů, které umožňují formulovat variační úlohu (3.16) lokálně bez požadavku na spojitost normálových stop bazických funkcí $\boldsymbol{\omega}_{K,F}$. Při použití prostorů nejnižšího řádu lze koeficienty $Z_{j,F}$ interpretovat jako střední hodnotu veličiny Z_j na nadstěně F.

Při označení $B_{i,j,K,E,F} = \int_K \boldsymbol{\omega}_{K,E}^T\mathbf{D}_{i,j}^{-1}\boldsymbol{\omega}_{K,F}$ můžeme rovnost (3.16) zapsat ve tvaru

$$\sum_{E\in\partial K} v_{i,j,K,E}B_{i,j,K,E,F} = Z_{j,K} - Z_{j,F}. \tag{3.18}$$

[4] Cell-Centered Finite Volume Method

Koeficienty $B_{i,j,K,E,F}$ tvoří na elementu K pozitivně definitní matici $\mathbf{B}_{i,j,K} = \left[B_{i,j,K,E,F}\right]_{E,F\in\mathcal{E}_K}$ a označme její inverzi $\mathbf{b}_{i,j,K} = \left[b_{i,j,K,E,F}\right]_{E,F\in\mathcal{E}_K}$. Konkrétní tvary matic $\mathbf{B}_{i,j,K}$ a $\mathbf{b}_{i,j,K}$ pro různé typy elementů jsou uvedeny v příloze A. Inverzí vztahu (3.18) získáme

$$v_{i,j,K,E} = b_{i,j,K,E} Z_{j,K} - \sum_{F\in\mathcal{E}_K} b_{i,j,K,E,F} Z_{j,F}, \tag{3.19}$$

kde $b_{i,j,K,E} = \sum\limits_{F\in\mathcal{E}_K} b_{i,j,K,E,F}$. Dále dodefinujeme $b_{i,j,K,E,F} = 0$ pro všechna $E, F \in \mathcal{E}_K$ pro případ, že $\mathbf{D}_{i,j}|_K = \mathbf{0}$. Pak již můžeme vyjádřit koeficienty $v_{i,K,E}$ pomocí $Z_{j,K}$ a $Z_{j,F}$:

$$v_{i,K,E} = \sum_{j=1}^{n} \left(b_{i,j,K,E} Z_{j,K} - \sum_{F\in\mathcal{E}_K} b_{i,j,K,E,F} Z_{j,F} \right) + w_{i,K,E}. \tag{3.20}$$

3.1.5 Diskretizace skalární rovnice

Variační formulaci úlohy (2.1) získáme vynásobením i-té rovnice (2.1) funkcí $\varphi \in V \equiv L^2(\Omega)$ a integrací přes oblast Ω:

$$\sum_{j=1}^{n} \int_\Omega N_{i,j} \frac{\partial Z_j}{\partial t}\varphi + \sum_{j=1}^{n} \int_\Omega \boldsymbol{u}_{i,j} \cdot \nabla Z_j \varphi + \int_\Omega \nabla \cdot \left(m_i \boldsymbol{v}_i + \sum_{j=1}^{n} Z_j \boldsymbol{a}_{i,j} \right) \varphi + \sum_{j=1}^{n} \int_\Omega r_{i,j} Z_j \varphi = \int_\Omega f_i \varphi. \tag{3.21}$$

Pro přibližné řešení této úlohy použijeme přístup nespojité Galerkinovy metody (DGM). Splnění rovnice (3.21) budeme požadovat na podprostoru $V_h \equiv D_0(\mathcal{K}_h)$. Dosazením $\varphi = \varphi_K$ a použitím vztahů (3.13) spolu s vlastnostmi bazických funkcí (3.12) dostaneme pro členy obsahující pouze skalární koeficienty následující náhrady:

$$\int_\Omega N_{i,j} \frac{\partial Z_j}{\partial t}\varphi_K \approx |K|_d N_{i,j,K} \frac{\mathrm{d}Z_{j,K}}{\mathrm{d}t}, \tag{3.22a}$$

$$\int_\Omega r_{i,j} Z_j \varphi_K \approx |K|_d r_{i,j,K} Z_{j,K}, \tag{3.22b}$$

$$\int_\Omega f_i \varphi_K \approx |K|_d f_{i,K}. \tag{3.22c}$$

Pro úpravu členů obsahující vektorové koeficienty použijeme vztahy (3.8), vlastnosti bazických funkcí (3.2) a (3.12) a Greenovu formuli:

$$\begin{aligned} \int_\Omega \boldsymbol{u}_{i,j} \cdot \nabla Z_j \varphi_K &= \int_{\partial K} Z_j \varphi_K \boldsymbol{u}_{i,j} \cdot \boldsymbol{n}_{\partial K} - \int_K Z_j \varphi_K \nabla \cdot \boldsymbol{u}_{i,j} - \underbrace{\int_K Z_j \boldsymbol{u}_{i,j} \cdot \nabla \varphi_K}_{=0} \\ &\approx \sum_{E\in\mathcal{E}_K} Z^{upw}_{i,j,E} u_{i,j,K,E} - Z_{j,K} \sum_{E\in\mathcal{E}_K} u_{i,j,K,E}, \end{aligned} \tag{3.23a}$$

$$\int_\Omega \nabla \cdot (m_i \boldsymbol{v}_i)\, \varphi_K = \int_{\partial K} m_i \varphi_K \boldsymbol{v}_i \cdot \boldsymbol{n}_{\partial K} - \underbrace{\int_K m_i \boldsymbol{v}_i \cdot \nabla \varphi_K}_{=0} \approx \sum_{E \in \mathcal{E}_K} m_{i,E}^{upw} v_{i,K,E}, \tag{3.23b}$$

$$\int_\Omega \nabla \cdot \left(Z_j \boldsymbol{a}_{i,j}\right) \varphi_K = \int_{\partial K} Z_j \varphi_K \boldsymbol{a}_{i,j} \cdot \boldsymbol{n}_{\partial K} - \underbrace{\int_K m_i \boldsymbol{v}_i \cdot \nabla \varphi_K}_{=0} \approx \sum_{E \in \mathcal{E}_K} Z_{i,j,E}^{upw} a_{i,j,K,E}, \tag{3.23c}$$

kde $\boldsymbol{n}_{K,E}$ označuje jednotkový vektor vnější normály na nadstěně E vzhledem k elementu K a symboly $m_{i,E}^{upw}(t)$ a $Z_{i,j,E}^{upw}(t)$ představují zatím blíže neurčené hodnoty veličin m_i a Z_j na nadstěně E, které jsou v případě vnitřní nadstěny společná pro oba elementy sousedící s nadstěnou E. Pro volbu těchto koeficientů použijeme techniku *upwind*, která bude popsána v sekci 3.3.

Použitím výše uvedených náhrad v rovnici (3.21) dostaneme následující semi-diskrétní schéma:

$$|K|_d \sum_{j=1}^{n} N_{i,j,K}(t) \frac{\mathrm{d}Z_{j,K}(t)}{\mathrm{d}t} + \sum_{j=1}^{n} \sum_{E \in \mathcal{E}_K} Z_{i,j,E}^{upw}(t) \left(a_{i,j,K,E}(t) + u_{i,j,K,E}(t)\right) + \sum_{E \in \mathcal{E}_K} m_{i,E}^{upw}(t) v_{i,K,E}(t)$$
$$+ \sum_{j=1}^{n} \left(|K|_d r_{i,j,K}(t) - \sum_{E \in \mathcal{E}_K} u_{i,j,K,E}(t) \right) Z_{j,K}(t) = |K|_d f_{i,K}(t). \tag{3.24}$$

Pro časovou diskretizaci schématu využijeme rozdělení intervalu $[0, T]$ na malé úseky s množinou hraničních bodů $\mathcal{T}$,

$$\mathcal{T} = \{0 = t_0 < t_1 < ... < t_M = T\}, \tag{3.25}$$

přičemž délky časových kroků označíme jako $\Delta t_k = t_{k+1} - t_k$. Dále budeme hodnoty časově závislých funkcí vyčíslených na časové hladině $t_k \in \mathcal{T}$ značit pomocí horního indexu k.

Nelineární semi-diskrétní schéma (3.24) diskretizujeme v čase pomocí zpětné Eulerovy metody a linearizujeme pomocí metody zamrzlých koeficientů. Tím získáme rovnice diskrétního schématu

$$\frac{|K|_d}{\Delta t_k} \sum_{j=1}^{n} N_{i,j,K} \left(Z_{j,K}^{k+1} - Z_{j,K}^{k} \right) + \sum_{j=1}^{n} \sum_{E \in \mathcal{E}_K} Z_{i,j,E}^{k,upw} \left(a_{i,j,K,E} + u_{i,j,K,E}\right) + \sum_{E \in \mathcal{E}_K} m_{i,E}^{k,upw} v_{i,K,E}^{k+1}$$
$$+ \sum_{j=1}^{n} \left(|K|_d r_{i,j,K} - \sum_{E \in \mathcal{E}_K} u_{i,j,K,E} \right) Z_{j,K}^{k+1} = |K|_d f_{i,K}, \tag{3.26}$$

platné pro všechna $i = 1, \ldots, n$, všechny elementy $K \in \mathcal{K}_h$ a časové hladiny $k = 0, \ldots, M-1$. Všechny koeficienty vystupující v rovnici (3.26) jsou vyčísleny na časové hladině t_k, pro přehlednost však vynecháváme horní index k u příslušných symbolů.

Hodnoty koeficientů $v_{i,K,E}$ máme vyjádřeny v (3.20), pro spojení s rovnicí (3.26) je však potřebujeme vyjádřit v čase $t = t_{k+1}$. Nelineární vztah (3.20) linearizujeme v čase opět pomocí metody zamrzlých koeficientů:

$$v_{i,K,E}^{k+1} = \sum_{j=1}^{n} \left(b_{i,j,K,E} Z_{j,K}^{k+1} - \sum_{F \in \mathcal{E}_K} b_{i,j,K,E,F} Z_{j,F}^{k+1} \right) + w_{i,K,E}, \tag{3.27}$$

kde hodnoty $b_{i,j,K,E}$, $b_{i,j,K,E,F}$ a $w_{i,K,E}$ jsou vyčísleny na časové hladině t_k.

3.2 Diskretizace okrajových podmínek

Při diskretizaci okrajových podmínek předpokládáme, že na všech vnějších nadstěnách $E \in \mathcal{E}_h^{ext}$ je dána podmínka právě jednoho typu, tedy pro všechna $j = 1, \ldots, n$ platí buď $E \subset \Gamma_j^{\mathcal{D}}$ nebo $E \subset \Gamma_j^{\mathcal{N}}$. Tento předpoklad lze chápat jako diskretizaci podmínky (2.5).

Na Dirichletově části hranice $\Gamma_j^{\mathcal{D}}$ dle (2.3a) předepisujeme přímo hodnotu funkce Z_j, diskretizací na okrajové nadstěně E získáme přímo střední hodnotu $Z_{j,E}^k$:

$$Z_{j,E}^k = \frac{1}{|E|_{d-1}} \int_E Z_j^{\mathcal{D}}(\boldsymbol{x}, t_k) \overset{\text{ozn}}{=} Z_{j,E}^{\mathcal{D}}(t_k). \tag{3.28}$$

Diskretizací Neumannovy podmínky (2.3b) na okrajové nadstěně $E \in \mathcal{E}_K$ získáme hodnotu koeficientu $v_{i,K,E}$, který má podle (3.9) význam toku funkce $\boldsymbol{v}_i$ přes nadstěnu E ve směru vnější normály:

$$v_{i,K,E}^k = \int_E v_i^{\mathcal{N}}(\boldsymbol{x}, t_k) \overset{\text{ozn}}{=} v_{i,E}^{\mathcal{N}}(t_k). \tag{3.29}$$

3.3 Upwind

Nyní již můžeme upřesnit volbu koeficientů $m_{i,E}^{k,upw}$ a $Z_{i,j,E}^{k,upw}$. Pro zajištění stability schématu pro advekční úlohy použijeme stabilizační techniku *upwind* [30]. V závislosti na směru numerického toku funkcí $\boldsymbol{v}_i$, resp. $\boldsymbol{a}_{i,j} + \boldsymbol{u}_{i,j}$ přes jednotlivé nadstěny sítě definujeme koeficienty $m_{i,E}^{k,upw}$, resp. $Z_{i,j,E}^{k,upw}$ vztahy

$$m_{i,E}^{k,upw} = \begin{cases} m_{i,K_1}^k & \text{pokud } v_{i,K_1,E}^k \geq 0, \\ m_{i,K_2}^k & \text{jinak,} \end{cases} \tag{3.30a}$$

$$Z_{i,j,E}^{k,upw} = \begin{cases} Z_{j,K_1}^k & \text{pokud } a_{i,j,K_1,E} + u_{i,j,K_1,E} \geq 0, \\ Z_{j,K_2}^k & \text{jinak,} \end{cases} \tag{3.30b}$$

pro všechny vnitřní nadstěny $E \in \mathcal{E}_h^{int}$, přičemž $E \in \mathcal{E}_{K_1} \cap \mathcal{E}_{K_2}$, a dále

$$m_{i,E}^{k,upw} = \begin{cases} m_{i,E}^{\mathcal{D}}(t_k) & \text{pokud } E \in \mathcal{E}_{h,i}^{ext,in}(t_k), \\ m_{i,K_1}^k & \text{jinak,} \end{cases} \tag{3.30c}$$

$$Z_{i,j,E}^{k,upw} = \begin{cases} Z_{j,E}^{\mathcal{D}}(t_k) & \text{pokud } a_{i,j,K_1,E} + u_{i,j,K_1,E} < 0, \\ Z_{j,K_1}^k & \text{jinak,} \end{cases} \tag{3.30d}$$

pro všechny vnější nadstěny $E \in \mathcal{E}_h^{ext}$, přičemž $E \in \mathcal{E}_{K_1}$. Předpokládáme, že hodnoty $m_{i,E}^{\mathcal{D}}$ a $Z_{j,E}^{\mathcal{D}}$ reprezentující předepsané stopy příslušných veličin na vnější nadstěně E jsou k dispozici, kdykoli jsou dle vztahů (3.30) potřeba.

Abstraktní množinu $\mathcal{E}_{h,i}^{ext,in}(t_k) \subset \mathcal{E}_h^{ext}$ vystupující v (3.30c) lze volit dle potřeb konkrétní úlohy. V nejjednodušším případě můžeme volit $\mathcal{E}_{h,i}^{ext,in}(t_k) = \{E \in \mathcal{E}_h^{ext} \mid v_{i,E}^{\mathcal{N}}(t_k) < 0\}$ analogicky vztahu (3.30d). V případě úlohy dvoufázového proudění je potřeba provázat volbu množiny $\mathcal{E}_{h,i}^{ext,in}$ a Dirichletovy podmínky pro mobility $m_i = m_i(S_{\alpha_i})$ pro obě fáze α_1 a α_2 tak, aby saturace S_{α_1} a S_{α_2} odpovídající mobilitám $m_{\alpha_1,E}^{k,upw}$ a $m_{\alpha_2,E}^{k,upw}$ byly konzistentní. Pokud například na nadstěně $E \in \mathcal{E}_h^{ext}$, $E \in \mathcal{E}_K$

předepisujeme tok nesmáčivé fáze $v_{n,E}^{\mathcal{N}} < 0$ a tok smáčivé fáze $v_{w,E}^{\mathcal{N}} = 0$, přičemž vtláčení typicky probíhá při dané saturaci $S_n = S_n^{in}$, je potřeba předepsat odpovídající mobilitu $m_{i,E}^{\mathcal{D}}$ pro obě fáze a také ji pro obě fáze použít k volbě $m_{i,E}^{k,upw}$. Proto pro úlohu dvoufázového proudění s koeficienty uvedenými v sekci 2.1 volíme $\mathcal{E}_{h,1}^{ext,in}(t_k) = \mathcal{E}_{h,2}^{ext,in}(t_k) = \{E \in \mathcal{E}_h^{ext} \mid v_{j,E}^{\mathcal{N}}(t_k) < 0 \text{ pro nějaké } j \in \{1,2\}\}$. Pro úlohu kompozičního proudění s koeficienty uvedenými v sekci 2.2 volíme $\mathcal{E}_{h,1}^{ext,in}(t_k)$ a $\mathcal{E}_{h,2}^{ext,in}(t_k)$ stejným způsobem jako pro dvoufázové proudění a $\mathcal{E}_{h,3}^{ext,in}(t_k) = \{E \in \mathcal{E}_h^{ext} \mid v_{3,E}^{\mathcal{N}}(t_k) < 0\}$.

3.4 Formulace diskrétní úlohy

Spojením rovnic (3.26) a (3.27) pro $i = 1, ..., n$ získáme soustavu $n \times \#\mathcal{K}_h$ lineárních rovnic pro $n \times (\#\mathcal{K}_h + \#\mathcal{E}_h)$ neznámých $Z_{j,K}^{k+1}$, $Z_{j,E}^{k+1}$, $j = 1, ..., n$. Tuto soustavu uzavřeme přidáním $n \times \#\mathcal{E}_h^{int}$ rovnic reprezentujících bilanční podmínky pro vnitřní nadstěny sítě a $n \times \#\mathcal{E}_h^{ext}$ rovnic reprezentujících okrajové podmínky (3.28) a (3.29). Za předpokladu nulových zdrojů na vnitřních nadstěnách sítě odpovídá standardnímu zákonu zachování hmoty lokální bilance toku $\boldsymbol{q}_i = m_i \boldsymbol{v}_i$, tedy rovnice

$$\sum_{\ell=1}^{2} m_{i,K_\ell,E}^{k} \left[\sum_{j=1}^{n} \left(b_{i,j,K,E} Z_{j,K}^{k+1} - \sum_{F \in \mathcal{E}_K} b_{i,j,K,E,F} Z_{j,F}^{k+1} \right) + w_{i,K,E} \right] = 0 \tag{3.31}$$

pro všechna $i = 1, \ldots, n$ a všechny nadstěny $E \in \mathcal{E}_h^{int}$, přičemž K_1 a K_2 jsou sousední elementy oddělené nadstěnou E. Problém nastává, pokud je mobilita m_i rovna nule, což může nastat například při modelování vícefázového proudění, kdy se příslušná fáze v daném místě nevyskytuje nebo zanikne. Pokud je v rovnici (3.31) $m_{i,K_\ell,E}^{k} = 0$ pro $\ell = 1, 2$, pak je výsledná soustava lineárních rovnic singulární. Proto ke konstrukci diskrétní úlohy použijeme modifikovaný postup inspirovaný pracemi [17, 18, 25] a navržený v článku [16] a mobilitu $m_{i,K_\ell,E}^{k}$ v rovnici (3.31) nahradíme společnou hodnotou pro oba elementy $m_{i,E}^{k,upw}$, tedy $m_{i,E}^{k,upw} = m_{i,K_\ell,E}^{k}$ pro $\ell = 1, 2$. To nám umožní mobilitu $m_{i,E}^{k,upw}$ eliminovat z rovnice (3.31) za předpokladu, že $m_{i,E}^{k,upw} > 0$. To nás vede k bilanci veličiny $\boldsymbol{v}_i$ namísto toku $\boldsymbol{q}_i$, kterou použijeme pro všechny hodnoty mobility, tedy včetně $m_{i,E}^{k,upw} = 0$. Výsledná bilanční rovnice je tedy

$$\sum_{\ell=1}^{2} \left[\sum_{j=1}^{n} \left(b_{i,j,K,E} Z_{j,K}^{k+1} - \sum_{F \in \mathcal{E}_K} b_{i,j,K,E,F} Z_{j,F}^{k+1} \right) + w_{i,K,E} \right] = 0 \tag{3.32}$$

pro všechna $i = 1, \ldots, n$ a všechny nadstěny $E \in \mathcal{E}_h^{int}$ oddělující elementy K_1 a K_2. Rovnici (3.32) lze získat dosazením vztahu (3.27) do (3.10a), v předchozích sekcích jsme se nezabývali diskretizací toku $\boldsymbol{q}_i$ s ohledem na výsledný tvar rovnice (3.32).

Spojením rovnic (3.26), (3.27), bilance rychlostí na vnitřních nadstěnách (3.32) a okrajových podmínek (3.28) a (3.29) pro $i = 1, ..., n$ tedy získáme soustavu $n \times (\#\mathcal{K}_h + \#\mathcal{E}_h)$ lineárních rovnic pro stejný počet neznámých $Z_{j,K}^{k+1}$, $Z_{j,E}^{k+1}$, $j = 1, ..., n$, kterou můžeme symbolicky zapsat v blokovém tvaru

$$\begin{pmatrix} \mathbf{Q} & -\mathbf{R} \\ \mathbf{S} & -\mathbf{T} \end{pmatrix} \begin{pmatrix} \boldsymbol{Z}_{\mathcal{K}_h}^{k+1} \\ \boldsymbol{Z}_{\mathcal{E}_h}^{k+1} \end{pmatrix} = \begin{pmatrix} \boldsymbol{G} \\ -\boldsymbol{H} \end{pmatrix}, \tag{3.33}$$

kde $\boldsymbol{Z}_{\mathcal{K}_h}^{k+1} = [\boldsymbol{Z}_K^{k+1}]_{K \in \mathcal{K}_h}$, $\boldsymbol{Z}_K^{k+1} = [Z_{j,K}^{k+1}]_{j=1}^{n}$, $\boldsymbol{Z}_{\mathcal{E}_h}^{k+1} = [\boldsymbol{Z}_F^{k+1}]_{F \in \mathcal{E}_h}$ a $\boldsymbol{Z}_F^{k+1} = [Z_{j,F}^{k+1}]_{j=1}^{n}$. Tato soustava je však pro řešení zbytečně velká – můžeme totiž využít její blokové struktury a dále snížit počet rovnic v úloze. Pokud bloky $\mathbf{Q}$, $\mathbf{R}$ a $\boldsymbol{G}$ reprezentují rovnici (3.26), potom $\mathbf{Q}$ je blokově diagonální

s velikostí bloků $n \times n$ a výpočtem $\boldsymbol{Z}_{\mathcal{K}_h}^{k+1} = \mathbf{Q}^{-1}(\boldsymbol{G} + \mathbf{R}\boldsymbol{Z}_{\mathcal{E}_h}^{k+1})$ můžeme neznámé $\boldsymbol{Z}_{\mathcal{K}_h}^{k+1}$ ze soustavy eliminovat.

Blok neznámých $\boldsymbol{Z}_K^{k+1}$ můžeme vyjádřit pomocí $\boldsymbol{Z}_F^{k+1}$ ve tvaru

$$\boldsymbol{Z}_K^{k+1} = \sum_{F \in \mathcal{E}_K} \mathbf{Q}_K^{-1}\mathbf{R}_{K,F}\boldsymbol{Z}_F^{k+1} + \mathbf{Q}_K^{-1}\boldsymbol{G}_K, \tag{3.34}$$

kde $\mathbf{Q}_K$ jsou bloky matice $\mathbf{Q}$, $\mathbf{R}_{K,F}$ jsou bloky matice $\mathbf{R}$ a $\boldsymbol{G}_K$ jsou bloky vektoru $\boldsymbol{G}$, přičemž dle rovnice (3.26) máme

$$[\mathbf{Q}_K]_{i,j} = \frac{|K|_d}{\Delta t_k} N_{i,j,K} - \sum_{E \in \mathcal{E}_K} u_{i,j,K,E} + \sum_{E \in \mathcal{E}_K} m_{i,E}^{k,upw} b_{i,j,K,E} + |K|_d r_{i,j,K}, \tag{3.35a}$$

$$[\mathbf{R}_{K,F}]_{i,j} = \sum_{E \in \mathcal{E}_K} m_{i,E}^{k,upw} b_{i,j,K,E,F}, \tag{3.35b}$$

$$\begin{aligned}[\boldsymbol{G}_K]_i = |K|_d f_{i,K} + \frac{|K|_d}{\Delta t_k} \sum_{j=1}^{n} N_{i,j,K} Z_{j,K}^k - \sum_{E \in \mathcal{E}_K} m_{i,E}^{k,upw} w_{i,K,E} \\ - \sum_{j=1}^{n} \sum_{E \in \mathcal{E}_K} Z_{i,j,E}^{k,upw} (a_{i,j,K,E} + u_{i,j,K,E}).\end{aligned} \tag{3.35c}$$

Pokud označíme analogicky $\mathbf{T}_{E,F}$ bloky matice $\mathbf{T}$, $\mathbf{S}_{E,K}$ bloky matice $\mathbf{S}$ a $\boldsymbol{H}_E$ bloky vektoru $\boldsymbol{H}$, pak dle rovnic (3.10a) a (3.27) pro bloky příslušné vnitřním nadstěnám $E \in \mathcal{E}_h^{int}$ platí

$$[\mathbf{T}_{E,F}]_{i,j} = \sum_{\substack{K \in \mathcal{K}_h \\ E,F \in \mathcal{E}_K}} b_{i,j,K,E,F}, \qquad \forall F \in \mathcal{E}_h, \tag{3.36a}$$

$$[\mathbf{S}_{E,K}]_{i,j} = b_{i,j,K,E} \qquad \forall K \in \mathcal{K}_h : E \in \mathcal{E}_K, \tag{3.36b}$$

$$[\boldsymbol{H}_E]_i = \sum_{\substack{K \in \mathcal{K}_h \\ E \in \mathcal{E}_K}} w_{i,K,E}. \tag{3.36c}$$

Pro vnější nadstěny $E \in \mathcal{E}_h^{ext}$ místo bilanční podmínky (3.10a) vystupují v soustavě (3.33) diskretizované okrajové podmínky. V případě Neumannovy podmínky (3.29) mají bloky $\mathbf{T}_{E,F}$ a $\mathbf{S}_{E,K}$ stejný tvar jako v případě vnitřních nadstěn a blok $\boldsymbol{H}_E$ navíc obsahuje předepsanou hodnotu toku $v_{i,E}^{\mathcal{N}}$. V případě Dirichletovy podmínky (3.28) je blok $\mathbf{S}_{E,K}$ nulový, na diagonále $\mathbf{T}_{E,F}$ je jednička a na pravé straně v bloku $\boldsymbol{H}_E$ předepsaná hodnota $Z_{i,E}^{\mathcal{D}}$.

Po eliminaci neznámých $\boldsymbol{Z}_{\mathcal{K}_h}^{k+1}$ v soustavě (3.33) dojdeme k soustavě

$$\mathbf{A}\boldsymbol{Z}_{\mathcal{E}_h}^{k+1} = \boldsymbol{b}, \tag{3.37}$$

kde $\mathbf{A} = \mathbf{T} - \mathbf{S}\mathbf{Q}^{-1}\mathbf{R}$ a $\boldsymbol{b} = \boldsymbol{H} + \mathbf{S}\mathbf{Q}^{-1}\boldsymbol{G}$. Neznámé v soustavě (3.37) nazýváme *stupni volnosti* (DOF[5]), počet neznámých je $N_{dof} = n \times \#\mathcal{E}_h$. Matice $\mathbf{A}$ je za uvedených předpokladů řídká a regulární. Řešení $\boldsymbol{Z}_{\mathcal{E}_h}^{k+1} = \mathbf{A}^{-1}\boldsymbol{b}$ spolu s explicitním vztahem (3.34) umožňuje získat všechny hodnoty potřebné pro přechod na následující časovou hladinu $t = t_{k+1}$.

[5] Degrees of Freedom

3.5 Algoritmus výpočtu

Soustava (3.37) spolu s explicitním vztahem (3.34) udávají, jak spočítat aproximaci řešení $\boldsymbol{Z}_{\mathcal{K}_h}^{k+1}$ v čase t_{k+1} pomocí aproximace $\boldsymbol{Z}_{\mathcal{K}_h}^{k}$ z předchozí časové hladiny t_k. Abychom získali aproximaci řešení v čase $t = T$ pomocí počátečního stavu v čase t_0, musíme iteračně určit aproximaci ve všech časových hladinách $t_k \in \mathcal{T}$. Podrobný algoritmus výpočtu numerického řešení pak vypadá takto:

Algoritmus 1

1. Zvolíme $h > 0$ a oblast $\Omega \subset \mathbb{R}^d$ pokryjeme konformní sítí $\mathcal{K}_h$. Časový interval diskretizujeme pomocí množiny $\mathcal{T}$.
2. Načteme vstupní data charakterizující úlohu, tj. data popisující počáteční a okrajové podmínky a koeficienty rovnice (2.1) v diskrétní podobě.
3. Položíme $k = 0$, $t = t_0 = 0$.
4. Inicializujeme $Z_{j,K}^0$ pro všechna $j = 1, \ldots, n$ a $K \in \mathcal{K}_h$ dle počáteční podmínky (2.2).
5. Inicializujeme koeficienty $v_{i,K,E}^0$ konstantou 0 pro všechna $i = 1, \ldots, n$, $K \in \mathcal{K}_h$ a $E \in \mathcal{E}_K$.
6. Dokud $t_k < T$, opakujeme následující kroky:
   - 6.1. Aktualizujeme koeficienty $N_{i,j,K}$, $u_{i,j,K,E}$, $m_{i,K}$, $w_{i,K,E}$, $a_{i,j,K,E}$, $r_{i,j,K}$, $f_{i,K}$ pro všechna $i, j = 1, \ldots, n$, $K \in \mathcal{K}_h$, $E \in \mathcal{E}_K$.
   - 6.2. Dle (3.30) aktualizujeme upwindované koeficienty $m_{i,E}^{upw}$ a $Z_{i,j,E}^{upw}$ pro všechna $i, j = 1, \ldots, n$, $E \in \mathcal{E}_h$.
   - 6.3. Aktualizujeme lokální matice $\mathbf{b}_{i,j,K}$ dané volbou bazických funkcí pro danou síť pro všechna $i, j = 1, \ldots, n$ a $K \in \mathcal{K}_h$.
   - 6.4. Aktualizujeme lokální matice $\mathbf{R}_{K,F}$ a vektory $\boldsymbol{G}_K$ pomocí vztahů (3.35b) a (3.35c) pro všechna $i, j = 1, \ldots, n$ a $K \in \mathcal{K}_h$, $F \in \mathcal{E}_K$.
   - 6.5. Sestavíme lokální matice $\mathbf{Q}_K$ dle vztahu (3.35a) a vypočteme řešení $\mathbf{Q}_K^{-1}\mathbf{R}_{K,F}$, $\mathbf{Q}_K^{-1}\boldsymbol{G}_K$ pro všechna $K \in \mathcal{K}_h$, $F \in \mathcal{E}_K$.
   - 6.6. Sestavíme matici $\mathbf{A}$ a vektor pravé strany $\boldsymbol{b}$ v soustavě (3.37).
   - 6.7. Řešíme soustavu (3.37) pro N_{dof} neznámých $Z_{i,E}^{k+1}$, $i = 1, \ldots, n$, $E \in \mathcal{E}_h$.
   - 6.8. Dle (3.34) vypočteme $Z_{i,K}^{k+1}$ pro všechna $i = 1, \ldots, n$, $K \in \mathcal{K}_h$.
   - 6.9. Aktualizujeme koeficienty $v_{i,K,E}^{k+1}$ dle vztahu (3.27) pro všechna $i = 1, \ldots, n$, $K \in \mathcal{K}_h$ a $E \in \mathcal{E}_K$.
   - 6.10. Přejdeme na následující časovou hladinu, $t_{k+1} = t_k + \Delta t_k$ a $k := k + 1$.
7. V čase $t_k = T$ získáme numerické řešení dané úlohy.

Kapitola 4

Implementace numerického schématu

V této kapitole se budeme věnovat implementaci numerického schématu pro řešení úlohy (2.1) popsaného v předchozí kapitole. Řešič pro uvedenou úlohu je implementován v jazyce C++ s využitím šablonové knihovny `TNL`,[1] kterou vyvíjíme na Katedře matematiky FJFI pod vedením Ing. Tomáše Oberhubera, Ph.D. Hlavní výhodou knihovny `TNL` je poskytnutí jednotného rozhraní pro implementaci hybridních algoritmů využívajících různé výpočetní architektury. Ve výsledném programu tak např. lze v závislosti na zvoleném parametru spustit výpočet buď na CPU nebo na GPU, což je velmi užitečné pro porovnání efektivity různých algoritmů pro stejnou úlohu. Pro výběr algoritmů jsou využity šablony jazyka C++, čímž odpadá dodatečná režie za běhu programu. Z knihovny `TNL` kromě tohoto rozhraní dále využíváme zejména numerické sítě, formáty pro reprezentaci řídkých matic a iterační algoritmy pro řešení soustav lineárních algebraických rovnic. Mezi autorovy příspěvky do knihovny `TNL` patří kromě implementace nových algoritmů a datových struktur popsaných v této kapitole také zejména optimalizace již existujících vnitřních struktur pro architekturu GPU (např. *shared pointers*) a podrobné testování.

Paralelní řešič je implementován pro architektury vícejádrových CPU s využitím OpenMP [10] a pro architektury moderních GPU s využitím platformy CUDA [37]. Standard OpenMP zavádí sadu direktiv kompilátoru pro správu vláken, což usnadňuje programování systémů se sdílenou pamětí. Podobně i v platformě CUDA jsou mapování vláken na výpočetní jednotky a správa paměti abstrahovány pomocí rozšíření jazyka C++. Ačkoli je z programátorského hlediska programování pro tyto architektury intuitivní a relativně jednoduché, vytvořit efektivní kód je v obou případech podstatně složitější, jelikož optimalizace zpravidla vyžadují podrobné znalostí dané hardwarové architektury.

Vývoj optimálního kódu je komplikovaný zejména z hlediska efektivního využití hierarchie pamětí a dobrého rozprostření řešené úlohy mezi výpočetní jednotky. Moderní vícejádrová CPU mohou mít nejvýše desítky nezávislých výpočetních jednotek pracujících na relativně vysoké frekvenci. Pro zajištění efektivní komunikace s pomalou operační pamětí jsou CPU vybavena relativně velkým množstvím rychlé vyrovnávací paměti (řádově desítky MiB). Naopak GPU mají typicky tisíce výpočetních jednotek, které jsou oproti moderním sofistikovaným CPU podstatně primitivnější a pracují na nižší frekvenci, ale jako celek poskytují mnohonásobně vyšší výpočetní výkon. GPU jsou dále vybaveny samostatnou globální pamětí, která je až řádově rychlejší oproti běžným modulům operační paměti typu DDR4. Výpočetní jednotky GPU jsou rozděleny do několika nezávislých bloků, tzv. *multiprocesorů*. Každý multiprocesor je dále vybaven velmi rychlou *sdílenou pamětí*, kterou lze využít ke komunikaci mezi vlákny v rámci bloku nebo jako ručně spravovanou cache.

V jednotlivých sekcích této kapitoly nejprve ukážeme důležitost rozložení dat v paměti a představíme flexibilní řešení tohoto problému v hybridním kódu určeném pro různé výpočetní architektury. Poté popíšeme paralelizaci algoritmu pro výpočet numerického řešení popsaného v sekci 3.5 a techniky pro zvýšení efektivity iterační metody GMRES[2] pro řešení soustav lineárních algebraických rovnic. V poslední sekci se budeme věnovat implementaci nestrukturovaných sítí v rámci knihovny `TNL`.

[1] Template Numerical Library

[2] Generalized Minimal Residual method

4.1 Rozložení dat v paměti

Koeficienty $N_{i,j,K}$, $b_{i,j,K,E,F}$, $R_{i,j,K,F}$ apod. vystupující v algoritmu 1 reprezentujeme v paměti jako prvky vícerozměrného pole odpovídající dimenze. Jelikož jazyk C++ nemá přímou podporu pro vícerozměrná pole, budeme muset navrhnout vlastní datovou strukturu a ručně provést mapování na jednorozměrné pole. Uvažme k-rozměrné pole $\mathcal{A}$ o velikosti dané multiindexem $A = (A_1, \ldots, A_k) \in \mathbb{N}^k$ a jednorozměrné pole $\mathcal{B}$ o velikosti $B = \Pi_{i=1}^{k} A_i$. Potom prvku pole $\mathcal{A}$ s multiindexem $\alpha = (\alpha_1, \ldots, \alpha_k) \in \mathbb{N}_0^k$, $\alpha < A$ můžeme jednoznačně přiřadit prvek pole $\mathcal{B}$ s indexem $\beta \equiv \beta_k$ pomocí rekurentního vztahu

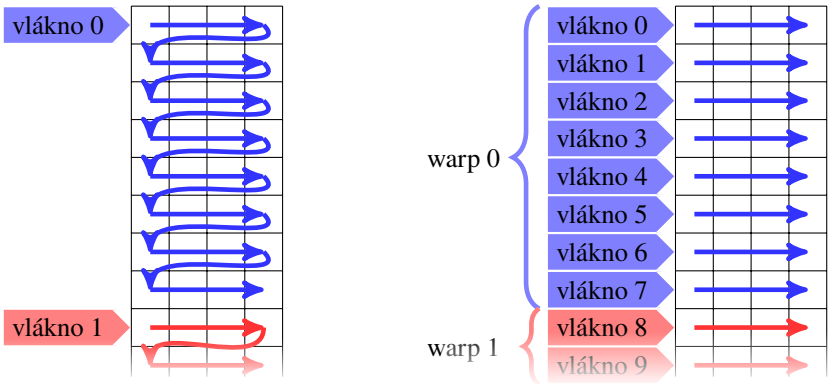
$$\beta_1 = \alpha_{\pi(1)}, \tag{4.1a}$$

$$\beta_j = \beta_{j-1} A_{\pi(j-1)} + \alpha_{\pi(j)}, \qquad \text{pro } j = 2, \ldots, k, \tag{4.1b}$$

kde π je nějaká permutace množiny $\{1, \ldots, k\}$. Při tomto číslování jsou prvky pole $\mathcal{A}$ číslovány postupně počínaje $\pi(k)$-tou složkou a konče $\pi(1)$-ní složkou indexu α.

Otázkou zůstává, jaké zvolit pořadí indexů pro koeficienty numerického schématu. V současných výpočetních architekturách zabírá přístup k datům v paměti významnou část výpočetního času, proto je potřeba optimalizovat datové struktury a algoritmy tak, aby bylo možné dosáhnout optimálního využití paměťového subsystému a výpočetních jednotek. Situaci komplikuje fakt, že různé výpočetní architektury mají různé požadavky na optimální rozložení dat v paměti. To ilustrujeme na příkladu dvourozměrného pole (viz obr. 4.1). Předpokládejme, že počet sloupců je malý oproti počtu řádků a pro zpracování elementární úlohy je potřeba přečíst data z celého řádku.

- Pro optimální využití CPU je třeba, aby přístup k datům byl sekvenční a aby každé vlákno zpracovávalo dostatečně velký blok dat. Počet sloupců je malý, tedy každé vlákno bude zpracovávat několik po sobě jdoucích řádků jako na obr. 4.1a. Každý řádek tedy představuje elementární úlohu pro nějaké vlákno, optimální číslování je tedy po řádcích.
- Pro optimální využití GPU je třeba spustit velký počet vláken, která na sobě nejsou zcela nezávislá. Vlákna jsou organizována do malých skupin nazývaných *warpy* a pro optimální využití výpočetních jednotek musí všechna vlákna ve stejném warpu vždy zpracovávat stejnou instrukci, jinak dojde k serializaci výpočtu a potenciálně mnohonásobnému zpomalení. Aby bylo možné rychle obsloužit požadavky na přístup do paměti pro celou skupinu vláken, je nutné zajistit, aby sousední vlákna přistupovala vždy do sousedních míst v paměti. V takovém případě může plánovač využít tzv.



(a) CPU (b) GPU

Obrázek 4.1: Typické zpracování dvourozměrného pole na CPU a GPU.

sloučených přístupů (anglicky *coalesced accesses*) a použít menší počet instrukcí, než když každé vlákno přistupuje jinam.

V našem příkladu na obr. 4.1b pro jednoduchost předpokládáme rozdělení vláken do warpů po 8 vláknech. Každé vlákno zpracovává jeden řádek stejně jako v případě CPU, ale skupina vláken přistupuje vždy do stejného sloupce pole, tedy optimální číslování je po sloupcích.

Vidíme, že v závislosti na výpočetní architektuře je potřeba provést optimalizaci přístupu k prvkům vícerozměrného pole spočívající v permutaci indexů ve vztahu (4.1a). Jedním z možných řešení je vytvořit specializace postiženého kódu podle výpočetní architektury a vždy ručně uvést pořadí indexů. To však vede ke značné duplikaci kódu, proto navrhneme obecnou datovou strukturu `NDArray`, která umožní konfigurovat permutace indexů staticky pomocí šablonových parametrů. Nejprve vysvětlíme rozhraní datové struktury na příkladu pole `R_ijKe` pro uložení koeficientů $R_{i,j,K,E}$ pro všechna $i, j = 1, \ldots, n$, $K \in \mathcal{K}_h$ a $E \in \mathcal{E}_K$, přičemž nadstěny $E \in \mathcal{E}_K$ indexujeme pomocí $e \in \{1, \ldots, \#\mathcal{E}_K\}$. [3] Vytvoření pole pak provedeme následovně:

```
1  NDArray< RealType,
2           SizesHolder< IndexType, n, n, 0, N_e >,  // i, j, K, e
3           std::index_sequence< 0, 2, 1, 3 >,  // i, K, j, e  (CPU)
4           std::index_sequence< 0, 1, 3, 2 >,  // i, j, e, K  (GPU)
5           DeviceType >
6       R_ijKe;
7  R_ijKe.setSizes( 0, 0, N_K, 0 );
8  // ...
9  R_ijKe( i, j, K, e ) = someValue;
```

Šablona `NDArray` má následující parametry:

- `RealType` určuje typ prvků pole, např. `float` nebo `double`.
- `SizesHolder< IndexType, n, n, 0, N_e >` určuje rozměry pole. Rozměry jsou typu `IndexType` a mohou být buď statické (známé již v době překladu) nebo dynamické (známé až po spuštění programu). Jelikož dynamické proměnné nelze použít jako šablonový parametr, použijeme `0` pro identifikaci dynamických složek a velikost zadáme později pomocí metody `setSize`. V našem případě vytváříme pole o velikosti $n \times n \times N_K \times N_e$, kde n je konstanta představující počet rovnic v soustavě (2.1), N_e je konstanta představující velikost množiny $\mathcal{E}_K$ pro daný typ sítě a N_K je velikost množiny $\mathcal{K}_h$, kterou určíme pro danou síť až za běhu programu. Statické rozměry pole umožní kompilátoru provést optimalizace pro výpočet výrazu (4.1a).
- `std::index_sequence< 0, 2, 1, 3 >` určuje pořadí indexů pro výpočet na CPU, v našem případě i, K, j, e.
- `std::index_sequence< 0, 1, 3, 2 >` určuje pořadí indexů pro výpočet na GPU, v našem případě i, j, e, K.
- `DeviceType` určuje typ výpočetní architektury, tedy buď CPU nebo GPU. V šabloně `NDArray` se pak použije buď první nebo druhá permutace indexů.

Permutace indexů pro přístup k prvku ve volání `R_ijKe( i, j, K, e )` odpovídá permutaci pro zadání velikosti v parametru `SizesHolder` a metodě `setSizes`. Tato permutace je nezávislá na permutacích určujících umístění prvků v paměti pro jednotlivé výpočetní architektury. V tom spočívá flexibilita tohoto návrhu, neboť pro změnu rozložení dat v paměti stačí změnit permutace indexů na pouhých dvou řádcích.

[3] V jazyce C++ číslujeme vždy od 0, tuto transformaci nechť si čtenář domyslí.

4.1.1 Reprezentace velikosti vícerozměrného pole

Nejprve popíšeme reprezentaci velikosti vícerozměrného pole pomocí šablony `SizesHolder`. Máme následující požadavky:

- možnost uvést libovolný počet indexů
- statické rozměry reprezentovat pomocí šablonových parametrů, dynamické proměnné pouze pro dynamické rozměry

Vzhledem k prvnímu požadavku budeme potřebovat šablony s proměnným počtem parametrů a trik s rekurzivním děděním popsaným níže. Nejprve ale zadefinujeme pomocný typ `DimT` pro reprezentaci statických velikostí a hloubky rekurze a pomocnou šablonu `IndexTag` pro generování typů parametrizovaných přirozenými čísly:

```
using DimT = std::size_t;

template< DimT v >
struct IndexTag
{
    static constexpr DimT value = v;
};
```

Naše implementace šablony `SizesHolder` vypadá takto:

```
template< typename Index,
          DimT... sizes >
class SizesHolder
    : public SizesHolderLayer< Index, sizes... >
{
    using BaseType = SizesHolderLayer< Index, sizes... >;
public:
    using IndexType = Index;
    static constexpr DimT numberOfLevels = sizeof...( sizes );

    template< DimT level >
    __cuda_callable__
    Index getSize() const {
        return BaseType::getSize( IndexTag< numberOfLevels - level - 1 >() );
    }

    template< DimT level >
    void setSize( Index size ) {
        BaseType::setSize( IndexTag< numberOfLevels - level - 1 >(), size );
    }

    template< typename... Sizes >
    void setSizes( Sizes&&... newSizes ) {
        setSizes< SizesHolder, 0 >( *this, std::forward< Sizes >( newSizes )... );
    }
};
```

Každá instance šablony `SizesHolder` dědí od zatím neimplementované šablony `SizesHolderLayer` se stejnými parametry. Rozhraní této šablony obsahuje metody `getSize` a `setSize`, které vytvoří tag pro

danou úroveň pomocí které kompilátor identifikuje správnou přetíženou metodu typu `BaseType` a předají své argumenty této metodě. Metoda `setSizes` volá pomocnou funkci `setSizes`, která rekurzivně volá metodu `setSize` se správnou úrovní pro každou proměnnou v posloupnosti `newSizes`.

Při implementaci šablony `SizesHolderLayer` využijeme trik s rekurzivním děděním: instance této šablony s $k + 1$ parametry dědí od instance té stejné šablony s k parametry a implementační šablony `SizeHolder` pro zbývající parametr.

```
template< typename Index,
          DimT currentSize,
          DimT... otherSizes >
class SizesHolderLayer
    : public SizesHolderLayer< Index, otherSizes... >,
      public SizeHolder< Index,
                         IndexTag< sizeof...( otherSizes ) >,  // LevelTag
                         currentSize >
{
    using BaseType = SizesHolderLayer< Index, otherSizes... >;
    using Layer = SizeHolder< Index,
                              IndexTag< sizeof...( otherSizes ) >,  // LevelTag
                              currentSize >;
protected:
    using BaseType::getSize;
    using BaseType::setSize;
    using Layer::getSize;
    using Layer::setSize;
};
```

Zde opět vytváříme tag pro identifikaci úrovně parametru `currentSize` a pomocí direktivy `using` zpřístupníme deklarace přetížených metod. Hodnota tagu odpovídá počtu zbývajících rozměrů, abychom nemuseli původní počet rozměrů udávat jako další parametr. Proto jsme v šabloně `SizesHolder` použili číslování úrovní pozpátku.

Pro zastavení rekurze dědění musíme implementovat samostatnou specializaci:

```
template< typename Index,
          DimT currentSize >
class SizesHolderLayer< Index, currentSize >
    : public SizeHolder< Index,
                         IndexTag< 0 >,  // LevelTag
                         currentSize >
{
    using Layer = SizeHolder< Index,
                              IndexTag< 0 >,  // LevelTag
                              currentSize >;
protected:
    using Layer::getSize;
    using Layer::setSize;
};
```

Nyní se dostáváme k implementaci šablony `SizeHolder`, kde adresujeme druhý požadavek týkající se statických a dynamických rozměrů pole. V obecné verzi šablony určené pro `size > 0` implementujeme chování pro statické rozměry, tedy prázdnou metodu `setSize` a `constexpr` metodu `getSize` vracející hodnotu šablonového parametru `size`:

```
template< typename Index,
          typename LevelTag,
          DimT size >
class SizeHolder
{
public:
    __cuda_callable__
    constexpr Index getSize( LevelTag ) const {
        return size;
    }

    void setSize( LevelTag, Index newSize ) {
        TNL_ASSERT( newSize == 0, );
    }
};
```

V částečné specializaci pro `size == 0` implementujeme privátní dynamický atribut `size` a přístupové metody `getSize` a `setSize`:

```
template< typename Index,
          typename LevelTag >
class SizeHolder< Index, LevelTag, 0 >
{
    Index size = 0;
public:
    __cuda_callable__
    Index getSize( LevelTag ) const {
        return size;
    }

    void setSize( LevelTag, Index size ) {
        TNL_ASSERT( size >= 0, );
        this->size = size;
    }
};
```

4.1.2 Mapování multiindexu

Pro mapování na jednorozměrné pole implementujeme pomocnou funkci `getStorageIndex`, která pro zadanou permutaci π, velikost pole $A \in \mathbb{N}^k$ a multiindex $\alpha \in \mathbb{N}_0^k$ vypočte výraz (4.1a).

```
template< typename Permutation, typename SizesHolder, typename... Indices >
__cuda_callable__
typename SizesHolder::IndexType
getStorageIndex( const SizesHolder& sizes, Indices&&... indices )
{
    return Indexer< SizesHolder, Permutation >::getIndex( sizes,
                                                          std::forward< Indices >( indices )... );
}
```

Vlastní implementace bude ve struktuře `Indexer`, protože šablony struktur a tříd lze na rozdíl od šablon funkcí částečně specializovat. V obecné implementaci šablony `Indexer` pouze určíme výchozí hodnotu parametru `level` určujícího hloubku rekurze:

```
template< typename SizesHolder,
          typename Permutation,
          DimT level = SizesHolder::numberOfLevels - 1 >
struct Indexer
{};
```

Dále implementujeme specializaci určenou pro `level > 0`, kde provedeme rekurzivní výpočet:

```
template< typename SizesHolder,
          DimT level,
          DimT... perm >
struct Indexer< SizesHolder, std::index_sequence< perm... >, level >
{
    using Permutation = std::index_sequence< perm... >;
    using Index = typename SizesHolder::IndexType;

    template< typename... Indices >
    __cuda_callable__
    static Index getIndex( const SizesHolder& sizes,
                           Indices&&... indices )
    {
        static constexpr DimT idx = get< level >( perm... );
        return get< idx >( std::forward< Indices >( indices )... ) +
               sizes.template getSize< idx >() *
               Indexer< SizesHolder, Permutation, level - 1 >::getIndex( sizes,
                ↪  std::forward< Indices >( indices )... );
    }
};
```

A na závěr specializaci pro `level == 0`, která pouze vrací hodnotu posledního indexu:

```
template< typename SizesHolder,
          DimT... perm >
struct Indexer< SizesHolder, std::index_sequence< perm... >, 0 >
{
    using Permutation = std::index_sequence< perm... >;
    using Index = typename SizesHolder::IndexType;

    template< typename... Indices >
    __cuda_callable__
    static Index getIndex( const SizesHolder& sizes,
                           Indices&&... indices )
    {
        static constexpr DimT idx = get< 0 >( perm... );
        return get< idx >( std::forward< Indices >( indices )... );
    }
};
```

Parametr `level` je o 1 menší než počet zatím nezpracovaných indexů, což odpovídá číslování úrovní v šabloně `SizesHolder`. Navíc je implementačně jednodušší použít pro rekurzi klesající posloupnost

končící konstantou `0`, protože hodnoty šablonových parametrů vystupujících v částečné specializaci nesmí záviset na typech nespecializovaných parametrů.

Díky specializaci šablony `Indexer` pro `std::index_sequence< perm... >` reprezentující permutaci indexů a šablonové metody s proměnným počtem parametrů `indices` máme dvě nezávislé posloupnosti, z nichž můžeme extrahovat argument s libovolným indexem pomocí obecné šablonové funkce `get`. Nejprve implementujeme specializaci funkce `get` pro `index == 0`:

```
template< DimT index, typename T, typename... Ts >
constexpr typename std::enable_if< index == 0, T >::type&&
get( T&& arg, Ts&&... args )
{
    return std::forward< T >( arg );
}
```

Zde návratový typ funkce závisí na lokálním typu v šabloně `std::enable_if`, který je dostupný pouze za předpokladu splnění podmínky `index == 0`. Toto použití principu SFINAE[4] [46] umožňuje kontrolovat, která přetížená funkce se má použít.

Obecná specializace funkce `get` pro `index > 0` vrací argument s indexem `index - 1` z posloupnosti argumentů `args`:

```
template< DimT index, typename T, typename... Ts >
constexpr typename std::enable_if< (index > 0) && index <= sizeof...( Ts ),
          typename std::tuple_element< index, std::tuple< T, Ts... > >::type >::type&&
get( T&& arg, Ts&&... args )
{
    using ReturnType = typename std::tuple_element< index, std::tuple< T, Ts... > >::type;
    return std::forward< ReturnType >( get< index-1 >( std::forward< Ts >( args )... ) );
}
```

Zde jsme použili utilitu `std::tuple_element` pro určení návratového typu, při použití jazyka C++14 by však bylo možné spoléhat na automatickou dedukci typu na základě příkazu `return`.

Na závěr je vhodné implementovat doplňkovou specializaci se `static_assert`, který vždy selže, abychom dostali čitelnější chybová hlášení při zadání špatného indexu:

```
template< long long index, typename T, typename... Ts >
constexpr typename std::enable_if< (index < 0) || (index > sizeof...( Ts )), T >::type
get( T&& arg, Ts&&... args )
{
    static_assert( index >= 0 && index <= sizeof...( Ts ),
                   "invalid index passed to the get function" );
    return arg;
}
```

4.1.3 Vícerozměrné pole

Nyní již můžeme přikročit k implementaci šablony `NDArray` reprezentující vícerozměrné pole. Šablona splňující návrh představený v úvodu sekce 4.1 vypadá následovně:

[4] Substitution failure is not an error

```
template< typename Real,
          typename SizesHolder,
          typename PermutationCPU = std::make_index_sequence< SizesHolder::numberOfLevels >,
          typename PermutationGPU = std::make_index_sequence< SizesHolder::numberOfLevels >,
          typename Device = TNL::Devices::Host,
          typename Index = typename SizesHolder::IndexType >
class NDArray
{
public:
    using StorageArray = TNL::Containers::Array< Real, Device, Index >;
    using Permutation = typename std::conditional<
                            std::is_same< Device, TNL::Devices::Host >::value,
                            PermutationCPU, PermutationGPU >::type;

    static constexpr DimT numberOfLevels = SizesHolder::numberOfLevels;
    static_assert( Permutation::size() == numberOfLevels, "wrong permutation" );

    template< typename... Sizes >
    bool setSizes( Sizes&&... sizes ) {
        static_assert( sizeof...( sizes ) == numberOfLevels, "wrong number of sizes" );
        this->sizes.setSizes( std::forward< Sizes >( sizes )... );
        return array.setSize( StorageSizeGetter< SizesHolder >::get( this->sizes ) );
    }

    template< typename... Indices >
    __cuda_callable__
    Real&
    operator()( Indices&&... indices ) {
        static_assert( sizeof...( indices ) == numberOfLevels, "wrong number of indices" );
        return array[ getStorageIndex< Permutation >( sizes,
                                                      std::forward< Indices >( indices )... ) ];
    }

protected:
    SizesHolder sizes;
    StorageArray array;
};
```

Pomocí šablony `std::make_index_sequence` vytváříme identickou permutaci, která slouží jako výchozí typ pro permutace na CPU i GPU. Výchozí typ pro parametr `Device` je `TNL::Devices::Host` reprezentující výpočet na CPU.

Jako úložiště slouží jednorozměrné pole typu `TNL::Containers::Array< Real, Device, Index >`, jehož implementaci lze nalézt v knihovně `TNL`. Dále s pomocí šablony `std::conditional` vybereme permutaci indexů, kterou budeme dále používat. Následuje implementace metod `setSizes` a `operator()` včetně kontrol délek posloupností pomocí šablon, které jsme popsali v předchozích sekcích. Zbývá doplnit pomocné šablony `setSizes` a `StorageSizeGetter`, jejich implementaci si ale jistě čtenář domyslí sám. Pro praktické využití by dále bylo možné doplnit další metody, např. pro přímý přístup k jednorozměrnému poli nebo nastavení všech prvků pole na zadanou hodnotu.

4.1.4 Vliv permutace indexů na efektivitu výpočtu

Vliv permutace indexů vícerozměrných polí na efektivitu výpočtu demonstrujeme pomocí výpočtu přibližného řešení úlohy (2.1) na ekvidistantní obdélníkové síti o rozměrech 400×100 elementů.

Nejprve provedeme výpočet na CPU s optimálními permutacemi indexů pro všechna pole, poté na GPU s permutacemi indexů optimálními pro CPU a nakonec výpočet na GPU s permutacemi indexů optimálními pro GPU. Výpočet na CPU probíhá sériově na jednom vláknu, paralelizace numerického schématu pro GPU je popsána v sekci 4.2. V tab. 4.1 jsou uvedeny výsledky pro jednotlivé kroky algoritmu 1 ovlivněné permutací indexů. Zejména řešení globální soustavy (krok 6.7) není zahrnuto, neboť na něj nemá permutace indexů vícerozměrných polí pro koeficienty žádný vliv. Z výsledků je patrné, že špatným rozložením dat v paměti došlo k $2\times$ až $6\times$ zpomalení v závislosti na rozměrech pole a typu algoritmu. Celkově je rozdíl mezi použitím optimálních permutací a neoptimálních permutací přibližně $4\times$ při zanedbání doby potřebné pro řešení globální soustavy.

Krok	Symbolický popis výpočtu	CPU	GPU (P1)		GPU (P2)		PSp
		CT[s]	CT[s]	GSp	CT[s]	GSp	
6.1	$\mathbf{N}$, $\boldsymbol{u}$, $\boldsymbol{m}$, $\mathbf{D}$, $\boldsymbol{w}$, $\boldsymbol{a}$, $\boldsymbol{r}$, $\boldsymbol{f}$	13,61	0,90	15,11	0,55	24,57	1,63
6.2	$m_{i,E}^{k,upw}$, $Z_{i,j,E}^{k,upw}$	10,35	1,25	8,26	0,36	28,69	3,47
6.3	$\mathbf{b}_{i,j,K}$	0,65	0,29	2,29	0,10	6,44	2,81
6.4	$\mathbf{R}_{K,F}$, $\boldsymbol{G}_K$	5,39	2,76	1,95	0,48	11,24	5,77
6.5	$\mathbf{Q}_K^{-1}\mathbf{R}_{K,F}$, $\mathbf{Q}_K^{-1}\boldsymbol{G}_K$	6,83	1,69	4,05	0,47	14,48	3,58
6.6	$\mathbf{A}$	24,97	8,46	2,95	1,71	14,60	4,94
6.8	$Z_{i,K}^{k+1}$	2,43	1,09	2,23	0,32	7,69	3,44
6.9	$v_{i,K,E}^{k+1}$	4,48	0,86	5,19	0,30	15,14	2,91
	Uvedené kroky celkem	68,70	17,30	3,97	4,29	16,02	4,03

Tabulka 4.1: Vliv permutace indexů vícerozměrného pole na efektivitu výpočtu na GPU. Porovnáváme výpočet na CPU s optimálními permutacemi, výpočet na GPU s permutacemi pro CPU (P1) a výpočet na GPU s optimálními permutacemi (P2). V jednotlivých sloupcích je uveden výpočetní čas (CT), urychlení výpočtu na GPU oproti CPU (GSp) a urychlení permutací P2 oproti P1 (PSp).

4.2 Paralelizace numerického schématu na GPU

Při paralelizaci algoritmu pro výpočet řešení dané úlohy je nejprve nutné rozdělit úlohu na co největší počet malých, na sobě nezávislých kroků, které lze zpracovat současně. Kroky algoritmu popsaného v sekci 3.5 jsou na sobě závislé a je tedy nutné je provést v uvedeném pořadí, u většiny ale můžeme využít nezávislosti skrytých elementárních kroků.

První tři kroky představují inicializaci, kterou provedeme sériovým zpracováním v konstantním čase. Inicializace jednotlivých koeficientů v krocích 4 a 5 jsou již nezávislé pro různé hodnoty uvedených indexů i, j, K, E. Hlavním problémem je paralelizace časové smyčky v kroku 6. Kroky 6.1 až 6.10 je nutné provádět v uvedeném pořadí, přechody na následující krok tedy představují hlavní synchronizační bariéry. V krocích 6.1, 6.2, 6.4, 6.8 a 6.9 probíhají výpočty dle explicitních vztahů, které jsou navzájem nezávislé pro různé hodnoty uvedených indexů i, j, K, E, a proto je snadné tyto kroky paralelizovat. Paralelizace zbývajících kroků 6.3, 6.5, 6.6 a 6.7 je komplikovanější, popíšeme je tedy podrobněji v následujících sekcích.

4.2.1 Lokální inverze

Pro výpočet prvků lokálních matic $\mathbf{b}_{i,j,K}$ v závislosti na typu elementu $K \in \mathcal{K}_h$ buď máme k dispozici přímo explicitní vztah, nebo je nutné nejprve sestavit matici $\mathbf{B}_{i,j,K}$ a algoritmicky provést její inverzi. V prvním případě je paralelizace snadná stejně jako v předešlých krocích algoritmu, ve druhém případě provádíme inverzi podobně jako v případě inverze matic $\mathbf{Q}_K$.

Krok 6.5 sestává z výpočtu prvků lokálních matic $\mathbf{Q}_K$ a následného výpočtu inverzí $\mathbf{Q}_K^{-1}\mathbf{R}_{K,F}$ a $\mathbf{Q}_K^{-1}\boldsymbol{G}_K$ pro všechny elementy $K \in \mathcal{K}_h$ a nadstěny $F \in \mathcal{E}_K$. Prvky všech matic $\mathbf{Q}_K$ můžeme vypočítat navzájem nezávisle, ale pro výpočet inverzí musíme pracovat s maticí $\mathbf{Q}_K$ jako celkem. Tyto dvě fáze můžeme oddělit globální synchronizační bariérou, nejprve masivně paralelně určit prvky matic $\mathbf{Q}_K$ a po synchronizaci vlákna příslušná stejnému elementu sloučit a vypočítat inverze $\mathbf{Q}_K^{-1}\mathbf{R}_{K,F}$ a $\mathbf{Q}_K^{-1}\boldsymbol{G}_K$. V praxi je však výhodnější využít toho, že matice $\mathbf{Q}_K$ jsou obvykle malé ($n \times n$) a navíc je nebudeme dále potřebovat, což nám umožní vyhnout se pomalému zápisu matic do globální paměti. Proto hned od začátku vytvoříme pouze jedno vlákno na každý element a v každém vlákně sériově sestavíme lokální matici $\mathbf{Q}_K$ uloženou ve sdílené paměti a vypočteme inverze. Výpočet inverzí $\mathbf{Q}_K^{-1}\mathbf{R}_{K,F}$, $\mathbf{Q}_K^{-1}\boldsymbol{G}_K$ je ekvivalentní řešení soustavy s maticí $\mathbf{Q}_K$ a několika pravými stranami, což provedeme pomocí LU rozkladu matice $\mathbf{Q}_K$.

4.2.2 Sestavení globální matice

Matice soustavy (3.37) je řídká a její rozměry jsou typicky velké, proto je pro její reprezentaci potřeba použít speciální formát. V této práci použijeme formát *sliced-ellpack* [34, 38], jehož implementace optimalizovaná pro GPU je dostupná v knihovně `TNL`.

Tradičním přístupem pro sestavení matice pocházející z diskretizace parciální diferenciální rovnice v sekvenčních algoritmech je procházet síť po elementech $K \in \mathcal{K}_h$, na každém elementu napočítat všechny lokální členy a přičíst je k příslušným prvkům matice. Paralelizace tohoto přístupu však vyžaduje provést obarvení sítě [3], aby bylo možné vyhnout se konfliktům mezi různými elementy sítě, které přispívají ke stejným prvkům matice. Obarvením sítě se množina elementů $\mathcal{K}_h$ rozpadne na několik komponent, které lze zpracovat paralelně, ale různé komponenty se musí zpracovat sekvenčně, což může vést k nízké efektivitě paralelizace pro středně velké úlohy.

Díky použití metody MHFEM však každý řádek matice $\mathbf{A}$ v soustavě (3.37) odpovídá nějaké nadstěně $E \in \mathcal{E}_h$ a nenulové prvky pocházejí z nadstěn $F \in \mathcal{E}_{K_1} \cup \mathcal{E}_{K_2}$, kde K_1 a K_2 jsou sousední elementy oddělené nadstěnou E. Díky tomu můžeme síť procházet po nadstěnách a vyhnout se konfliktům i bez obarvení sítě. To je jedna z hlavních výhod numerického schématu, která zajišťuje dobrou efektivitu výpočtu na GPU. Navíc počet nenulových prvků pro každý řádek je pevně daný a závisí pouze na geometrii elementů sítě, což je výhodné pro efektivní zpracování matice na GPU, protože do matice není třeba vkládat nuly kvůli zarovnání a při násobení matice a vektoru nedochází k divergenci vláken. Pro sítě sestávající z jediného typu elementů, tedy s konstantním počtem nadstěn elementu $\#\mathcal{E}_K$, je počet nenulových prvků $n(2\#\mathcal{E}_K - 1)$, $n\#\mathcal{E}_K$, resp. 1 pro řádky matice $\mathbf{A}$ odpovídající vnitřním nadstěnám, nadstěnám s Neumannovou podmínkou, resp. nadstěnám s Dirichletovou podmínkou.

4.3 Řešení soustav lineárních algebraických rovnic s řídkou maticí

Řešení soustav lineárních algebraických rovnic s řídkou maticí $\mathbf{A} \in \mathbb{R}^{N,N}$ v kroku 6.7 představuje výpočetně nejnáročnější část algoritmu 1, což bylo ukázáno již v předchozích pracech autora [28, 29]. Podíl jednotlivých kroků na celkovém výpočetním čase závisí na konkrétní úloze, velikosti sítě a volbě časového kroku, v úlohách popsaných v kapitole 5 zabírá krok 6.7 přibližně 99% celkové doby výpočtu.

Řešení soustav lineárních algebraických rovnic je také implementačně nejsložitější část algoritmu 1. Přímé metody lze obecně velmi těžko paralelizovat a dobré škálování na velký počet výpočetních jednotek lze zajistit jen pro velmi velké úlohy. Pro řídké matice s komplikovanou strukturou navíc při faktorizaci vzniká zaplnění, které značně zvyšuje paměťové nároky. Přímé metody pro řešení soustav lineárních algebraických rovnic s obecnou řídkou maticí jsou implementovány např. v balících `UMFPACK` [12], `SuperLU` [31], `PARDISO` [40, 41] a `PaStiX` [24], které však nevyužívají akcelerátory GPU vůbec, nebo jen velmi omezeně. Z výše uvedených důvodů spoléháme v knihovně `TNL` na iterační metody, které mají konstantní paměťové nároky a jejich implementace i paralelizace je jednodušší. V této práci se budeme zabývat metodou GMRES [42, 43], což je jedna z nejpoužívanějších iteračních metod pro řešení soustav s obecnou maticí. V následující sekci ukážeme, jak metodu GMRES efektivně paralelizovat pro GPU.

4.3.1 Metoda CWYGMRES

Metoda GMRES je založena na Arnoldiho algoritmu pro konstrukci ortonormální báze Krylovova podprostoru $\mathcal{K}_s = \mathrm{span}\{\bar{\boldsymbol{v}}_1, \mathbf{A}\bar{\boldsymbol{v}}_1, \ldots, \mathbf{A}^{s-1}\bar{\boldsymbol{v}}_1\}$, jehož standardní podobu přebíráme z [42, str. 148]:

Algoritmus 2

1. Zvol $\bar{\boldsymbol{v}}_1 \in \mathbb{R}^N$ tak, že $\|\bar{\boldsymbol{v}}_1\| = 1$.
2. Pro $i = 1, \ldots, s$:
2.1. $\bar{\boldsymbol{w}}_i \coloneqq \mathbf{A}\bar{\boldsymbol{v}}_i$
2.2. Pro $k = 1, \ldots, i$:
2.2.1. $h_{ki} = \bar{\boldsymbol{w}}_i^T \bar{\boldsymbol{v}}_k$
2.2.2. $\bar{\boldsymbol{w}}_i \coloneqq \bar{\boldsymbol{w}}_i - h_{ki}\bar{\boldsymbol{v}}_k$
2.3. $h_{i+1,i} = \|\bar{\boldsymbol{w}}_i\|$. Pokud $h_{i+1,i} = 0$, skonči.
2.4. $\bar{\boldsymbol{v}}_{i+1} = \frac{1}{h_{i+1,i}}\bar{\boldsymbol{w}}_i$

Algoritmus 2 využívá v kroku 2.2 modifikovaný Gramův–Schmidtův ortogonalizační proces, což je ze své podstaty čistě sekvenční algoritmus, jelikož pořadí prováděných kroků zajišťuje numerickou stabilitu. V praxi ani přesnost algoritmu 2 nemusí být dostatečná a je potřeba provést reortogonalizaci opakováním kroku 2.2 dvakrát [22]. Algoritmus 2 však lze paralelizovat implicitně použitím paralelních algoritmů pro výpočet základních operací lineární algebry, jako např. součet vektorů nebo skalární součin.

Alternativním řešením je použití jiného ortogonalizačního procesu, který by mohl být vhodnější pro paralelizaci. Nahrazení modifikovaného Gramova–Schmidtova procesu v algoritmu 2 ortogonalizací pomocí Householderových transformací bylo poprvé navrženo v článku [47]. Touto modifikací získáme algoritmus 3, který opět přebíráme z [42, str. 149]:

Algoritmus 3

1. Zvol nenulový vektor $\boldsymbol{z}_1 \in \mathbb{R}^N$.
2. Pro $i = 1, \ldots, s+1$:
2.1. Urči Householderův vektor $\boldsymbol{y}_i \in \mathbb{R}^N$ takový, že
$(\boldsymbol{y}_i)_j = 0$ pro $j = 1, \ldots, i-1$ a $(\mathbf{P}_i\boldsymbol{z}_i)_j = 0$ pro $j = i+1, \ldots, N$, kde $\mathbf{P}_i = \mathbf{I} - \bar{t}_i\boldsymbol{y}_i\boldsymbol{y}_i^T$,
$\bar{t}_i = \dfrac{2}{\|\boldsymbol{y}_i\|^2}$.
2.2. $\boldsymbol{h}_{i-1} = \left[(\mathbf{P}_i\boldsymbol{z}_i)_j\right]_{j=1}^{s+1}$
2.3. $\bar{\boldsymbol{v}}_i = \mathbf{P}_1 \ldots \mathbf{P}_i\boldsymbol{e}_i$
2.4. Pokud $i \leq s$, vypočti $\boldsymbol{z}_{i+1} = \mathbf{P}_i \ldots \mathbf{P}_1\mathbf{A}\bar{\boldsymbol{v}}_i$.

Z hlediska paralelizace jsme si zatím nijak nepomohli, jelikož Householderovy transformace v krocích 2.3 a 2.4 algoritmu 3 je třeba aplikovat sekvenčně. Můžeme však využít kompaktní WY reprezentaci [44] součinu Householderových matic, která byla v nedávné době aplikována v paralelních algoritmech pro moderní hardware [27, 50]. Podobná technika byla navržena také v [47].

Pro modifikaci algoritmu 3 pomocí kompaktní WY reprezentace označme $\mathbf{Y}_i = [\boldsymbol{y}_1, \ldots, \boldsymbol{y}_i] \in \mathbb{R}^{N,i}$, $\mathbf{T}_1 = \bar{t}_1 \in \mathbb{R}^{1,1}$ a rekurzivně definujme horní trojúhelníkovou matici $\mathbf{T}_i \in \mathbb{R}^{i,i}$,

$$\mathbf{T}_i = \begin{pmatrix} \mathbf{T}_{i-1} & -\bar{t}_i \mathbf{T}_{i-1} \mathbf{Y}_{i-1} \boldsymbol{y}_i \\ 0 & \bar{t}_i \end{pmatrix}. \tag{4.2}$$

Součiny Householderových matic $\mathbf{P}_1 \ldots \mathbf{P}_i$ a $\mathbf{P}_i \ldots \mathbf{P}_1$ lze vyjádřit pomocí matic $\mathbf{Y}_i$ a $\mathbf{T}_i$ následujícím způsobem:

$$\mathbf{P}_1 \ldots \mathbf{P}_i = \mathbf{I} - \mathbf{Y}_i \mathbf{T}_i \mathbf{Y}_i^T, \tag{4.3a}$$

$$\mathbf{P}_i \ldots \mathbf{P}_1 = \mathbf{I} - \mathbf{Y}_i \mathbf{T}_i^T \mathbf{Y}_i^T. \tag{4.3b}$$

Tím dostáváme následující algoritmus:

Algoritmus 4

1. Zvol nenulový vektor $\boldsymbol{z}_1 \in \mathbb{R}^N$.
2. Pro $i = 1, \ldots, s+1$:

2.1. Pro $\boldsymbol{z}_i$ vypočti $\boldsymbol{y}_i$ a $\bar{t}_i$ stejně jako v algoritmu 3.

2.2. Aktualizuj matice $\mathbf{Y}_i$ a $\mathbf{T}_i$ pomocí $\bar{t}_i$, $\boldsymbol{y}_i$, $\mathbf{T}_{i-1}$ a $\mathbf{Y}_{i-1}$.

2.3. $\boldsymbol{h}_{i-1} = \left[(\mathbf{P}_i \boldsymbol{z}_i)_j\right]_{j=1}^{s+1}$

2.4. $\bar{\boldsymbol{v}}_i = \left(\mathbf{I} - \mathbf{Y}_i \mathbf{T}_i \mathbf{Y}_i^T\right) \boldsymbol{e}_i$

2.5. Pokud $i \leq s$, vypočti $\boldsymbol{z}_{i+1} = \left(\mathbf{I} - \mathbf{Y}_i \mathbf{T}_i^T \mathbf{Y}_i^T\right) \mathbf{A} \bar{\boldsymbol{v}}_i$.

Výhoda algoritmu 4 oproti algoritmu 3 spočívá v nahrazení sekvenční aplikace Householderových transformací výpočtem pevného počtu součinů plné matice a vektoru, což je operace, kterou lze dobře paralelizovat. Srovnání podstatných vlastnosti uvedených algoritmů je uvedeno v tab. 4.2. Z hlediska paměťové náročnosti ortogonalizace pomocí Householderových transformací a kompaktní WY reprezentace ve skutečnosti není potřeba ukládat vektory $\bar{\boldsymbol{v}}_i$, které lze dopočítat dle [42, str. 159], čímž dostaneme náročnost $O((s+1)N)$. To je však také čistě sekvenční algoritmus, navíc v praxi je škoda nevyužít dostatek operační paměti a globální paměti GPU.

V praxi provádíme výpočty v krocích 2.4 a 2.5 algoritmu 4 „zprava doleva“, aby se co nejrychleji redukoval objem zpracovávaných dat. Při výpočtu na GPU rozlišujeme výpočty vyžadující $O(N)$ operací, které provádíme na GPU, a výpočty vyžadující $O(s)$ nebo $O(s^2)$ operací, které provádíme sekvenčně na CPU. Aby to bylo možné, ukládáme matice $\mathbf{Y}_i$ v globální paměti GPU a matice $\mathbf{T}_i$ v operační paměti systému. V krocích 2.3 a 2.4 se nám však hodí přistupovat k prvkům matice $\mathbf{Y}_i$ při výpočtech na CPU, proto pro umožnění rychlého přístupu duplikujeme prvních $s+1$ řádků matic $\mathbf{Y}_i$ také v operační paměti.

Srovnání ortogonalizace pomocí modifikovaného Gramova–Schmidtova procesu s reortogonalizací (MGSR) a kompaktní WY reprezentace (CWY) lze nalézt v následující sekci v tab. 4.3 pro zkrácenou testovací úlohu a v tab. 5.4 a 5.5 pro celou numerickou simulaci popsanou v sekci 5.4.

	MGS	HO	CWY
Počet operací	$O(2s^2N)$	$O(4s^2N)$	$O(4s^2N)$
Počet synchronizací	$O(s^2)$	$O(s^2)$	$O(s)$
Paměťové nároky	$O((s+1)N)$	$O(2(s+1)N)$	$O(2(s+1)N)$
Ztráta ortogonality	$O(\epsilon\kappa(\mathbf{A}))$	$O(\epsilon)$	$O(\epsilon)$

Tabulka 4.2: Srovnání podstatných vlastnosti různých ortogonalizačních algoritmů: MGS označuje modifikovaný Gramův–Schmidtův proces, HO označuje Householderovy transformace a CWY označuje kompaktní WY reprezentaci. V posledním řádku tabulky ϵ označuje strojovou přesnost a $\kappa(\mathbf{A})$ označuje číslo podmíněnosti matice $\mathbf{A}$. Údaje přebíráme z [27, 42].

4.3.2 Adaptivní restartování metody GMRES

Je známo [42], že volba restartovacího parametru s má významný vliv na rychlost konvergence metody GMRES(s) z hlediska výpočetního času. Počet operací v jednom cyklu před restartováním je úměrný s^2, proto je běžnou snahou volit s co nejmenší. Pro úlohy se špatně podmíněnou maticí však může být pro malé s potřeba velmi velký počet iterací, který lze snížit zvětšením parametru s. V některých případech tak může být nutné volit parametr s dostatečně velký, aby byla zajištěna konvergence metody.

Pro volbu restartovacího parametru s metody GMRES(s) použijeme adaptivní strategii představenou v článku [1]. Myšlenka tohoto postupu spočívá v narušení opakujícího se chování metody s konstantním parametrem s a ve snaze použít co nejmenší hodnotu restartovacího parametru při rychlé konvergenci. Modifikovanou verzi tohoto postupu uvádíme v následujícím algoritmu:

Algoritmus 5

1. Vstupní parametry: $s_{min}, s_{max}, d_{min}, d_{max} \in \mathbb{N}$
2. Nastav $\beta = 1$, $\beta_{min} = 0{,}175$, $\beta_{max} = 0{,}99$, $i = 0$.
3. Dokud metoda nezkonvergovala:

3.1. Pokud $\beta > \beta_{max}$, zvol $s_i = s_{max}$.

3.2. Pokud $\beta < \beta_{min}$, zvol $s_i = s_{i-1}$.

3.3. Pokud $\beta \in [\beta_{min}, \beta_{max}]$, zvol $d = d_{min} + \left\lfloor \frac{d_{max}-d_{min}}{s_{max}-s_{min}}(s_{i-1} - s_{min}) \right\rfloor$ a pokud $s_{i-1} - d \geq s_{min}$, zvol $s_i = s_{i-1} - d$, jinak $s_i = s_{max}$.

3.4. Proveď nejvýše s_i iterací metody GMRES, skonči pokud bylo splněno kritérium konvergence.

3.5. Nastav $\beta = \frac{\|\boldsymbol{r}_{i+1}\|}{\|\boldsymbol{r}_i\|}$, kde $\boldsymbol{r}_i$ a $\boldsymbol{r}_{i+1}$ jsou rezidua před a po provedení iterací metody GMRES.

3.6. $i \coloneqq i + 1$

V algoritmu 5 je parametr s je volen podle rychlosti konvergence β. V prvním cyklu, nebo pokud v předchozím cyklu došlo k malému zmenšení normy rezidua, volíme $s = s_{max}$. Pokud naopak v předchozím cyklu došlo k výraznému poklesu normy rezidua, ponecháme hodnotu s beze změny. V ostatních případech postupně zmenšujeme hodnotu s v mezích s_{min} a s_{max} a po dosažení dolní meze skočíme opět na horní mez, čímž dochází k narušení opakujícího se chování konvergence metody GMRES.

Naše modifikace v algoritmu 5 oproti originálnímu postupu z článku [1] spočívá v zavedení parametrů d_{min} a d_{max} a volbě délky adaptivního kroku d pomocí lineární interpolace v závislosti na aktuální hodnotě s. Volba $d_{min} = d_{max}$ přesně odpovídá originálnímu algoritmu z [1] s konstantní délkou kroku d,

kterou autoři volí jako $d = 3$. Jemnější volba délky kroku d umožňuje rychleji přejít s parametrem s z okolí s_{max} do okolí s_{min} a tam provést více iterací, které trvají kratší dobu.

Přínos adaptivní volby parametru s demonstrujeme na zkrácené testovací úloze dvoufázového proudění ze sekce 5.4, jejíž výsledky pro obdélníkovou síť 120×120 jsou uvedeny v tab. 4.3. Oproti restartování po konstantním počtu kroků došlo při použití adaptivního restartování s konstantním krokem (algoritmus z článku [1]) ke zrychlení o přibližně 45%, resp. 35% v případě varianty MGSR, resp. CWY. Při použití adaptivního restartování s proměnným krokem (algoritmus 5) došlo ke zrychlení o přibližně 49%, resp. 39% oproti konstantnímu restartování v případě varianty MGSR, resp. CWY. Použité parametry $d_{min} = 2, d_{max} = 4$ vylepšují výsledky urychlení dosažené s parametry $d_{min} = d_{max} = 3$ použitými v [1], tedy zobecnění volby parametru d v algoritmu 5 má nezanedbatelný přínos na rychlost výpočtu zkoumané úlohy. Podobný přínos adaptivní volby restartovacího parametru pozorujeme i na dalších úlohách řešených algoritmem 1.

	konstantní restartování ($s = 30$)		adaptivní restartování s konstantním krokem ($s_{min} = 3, s_{max} = 30, d_{min} = 3, d_{max} = 3$)		adaptivní restartování s proměnným krokem ($s_{min} = 3, s_{max} = 30, d_{min} = 2, d_{max} = 4$)	
	MGSR	CWY	MGSR	CWY	MGSR	CWY
doba výpočtu $[s]$	633	273	347	177	318	165
počet iterací	616589	638187	428638	451477	403688	425943

Tabulka 4.3: Srovnání doby výpočtu a počtu iterací pro různé varianty metody GMRES(s): ortogonalizace buď pomocí modifikovaného Gramova–Schmidtova procesu s reortogonalizací (MGSR) nebo pomocí kompaktní WY reprezentace (CWY) a různé varianty pro volbu restartovacího parametru s. Výsledky odpovídají zkrácené testovací úloze dvoufázového proudění ze sekce 5.4 a byly vypočteny ve 2D na obdélníkové síti 120×120 (231360 stupňů volnosti) na GPU Nvidia Tesla K40.

4.3.3 Předpodmínění

Konvergenci iteračních metod pro řešení soustav lineárních algebraických rovnic je obecně možné urychlit použitím vhodného předpodmínění. Vývoj předpodmínění pro paralelní architektury je však komplikovaná úloha. V rámci knihovny `TNL` máme k dispozici pouze Jacobiho předpodmínění, pokročilejší metody budou předmětem dalšího výzkumu.

4.4 Nestrukturované sítě

V numerických simulacích využívajících nestrukturované sítě se typicky pracuje buď s obecnou reprezentací sítě z nějaké víceúčelové knihovny (např. `VTK` [45]), nebo s reprezentací navrženou konkrétně pro účely dané simulace nebo schématu (např. `Clawpack` [9]). Výhodou obecných reprezentací je množství funkcí pro popis velkého počtu úloh, naproti tomu výhodou specifických reprezentací je jednodušší návrh a přizpůsobení pro efektivní popis dané úlohy. Návrh rozhraní pro nestrukturované sítě v knihovně `TNL` kombinuje výhody obou přístupů. Rozhraní je navrženo obecně, ale pomocí statické konfigurace a šablon jazyka C++ lze optimalizovat jednotlivé *specializace* sítě pro potřeby konkrétních úloh.

Statická konfigurace nestrukturovaných sítí v `TNL` umožňuje měnit následující parametry:

- Topologie sítě určená typem elementů. Podporovány jsou pouze sítě, kde všechny elementy mají stejný typ. Typy entit nižších dimenzí jsou určeny typem entit vyšší dimenze.
- Dimenze prostoru, do kterého je síť vnořená. Musí být větší nebo rovná dimenzi elementů sítě.
- Datové typy souřadnic (např. `float`, `double`), globálních indexů a lokálních indexů (např. `int` a `short int`).
- Dimenze entit, které jsou v síti uloženy. Entity nejnižší a nejvyšší dimenze (vrcholy a elementy) musí být uloženy vždy.
- Kombinace dimenzí d_1, d_2, kde $d_1 > d_2$, pro které entity dimenze d_1 ukládají indexy svých subentit dimenze d_2.
- Kombinace dimenzí d_1, d_2, kde $d_1 < d_2$, pro které entity dimenze d_1 ukládají indexy okolních entit dimenze d_2.
- Dimenze entit, pro které se ukládá, zda je daná entita vnitřní nebo okrajová.

Jednotlivé parametry statické konfigurace ovlivňují jak velikost celé sítě, tak velikost jednotlivých entit a nepřímo i efektivitu využití pamětí cache a výpočetních jednotek. Při optimalizaci sítě pro danou úlohu můžeme například vynechat nepotřebné entity, indexy k sousedním entitám apod.

Implementace nestrukturovaných sítí splňující výše uvedené vlastnosti spoléhá na pokročilé metaprogramovací techniky využívající šablony jazyka C++, podobně jako implementace vícerozměrného pole popsaná v sekci 4.1. Struktura kódu je však značně komplikovaná a dokonce ani některé kompilátory jej nedokáží správně zpracovat, ačkoli se jedná o platný kód v jazyce C++11. Proto například pro překlad kódu pro GPU, jehož optimalizace popíšeme v následující sekci, nelze použít standardní překladač `nvcc` pro jazyk CUDA od společnosti Nvidia. Alternativní kompilátor pro jazyk CUDA plně kompatibilní se standardy jazyka C++11 a C++14 je `clang` [48], jehož verze 4.0 umožnila vývoj kódu pro nestrukturované sítě na GPU a tím splnit třetí bod zadání této diplomové práce s kladným výsledkem.

4.4.1 Optimalizace na GPU

Pro práci s nestrukturovanou sítí na GPU je potřeba provést sekvenční inicializaci na CPU a výsledek překopírovat na GPU. Na GPU pak paralelně probíhají pouze pomocné metody pro zajištění integrity sítě a metody na procházení jednotlivých entit sítě. Na rozdíl od strukturovaných ortogonálních sítí jsou entity všech dimenzí organizovány do lineárních datových struktur, pouze pro rychlé odlišení vnitřních a okrajových entit jsou potřeba dodatečná indexová pole.

Z návrhu je patrné, že s každou entitou sítě je propojeno poměrně velké množství informace. Pokud bychom všechny indexy na okolní entity ukládali přímo ve struktuře reprezentující danou entitu, byl by přístup k jednotlivým položkám této struktury na GPU značně neefektivní, protože by nebylo možné využít sloučených přístupů do globální paměti GPU. Podobně jako v případě vícerozměrného pole popsaného v sekci 4.1 potřebujeme provést transformaci dat v paměti, aby byly stejné položky pro entity zpracovávané sousedními vlákny uloženy v paměti vedle sebe. Proto indexy na okolní entity ukládáme v samostatných datových strukturách pro příslušné dimenze a ve strukturách reprezentující entity sítě máme pouze ukazatele na data v samostatné struktuře příslušející dané entitě. Díky tomu jsou velikosti struktur reprezentujících entity sítě malé, nezávisí na počtu okolních entit a při vhodném očíslování entit sítě by mělo být možné využít sloučených přístupů do globální paměti GPU.

4.4.2 Očíslování entit sítě

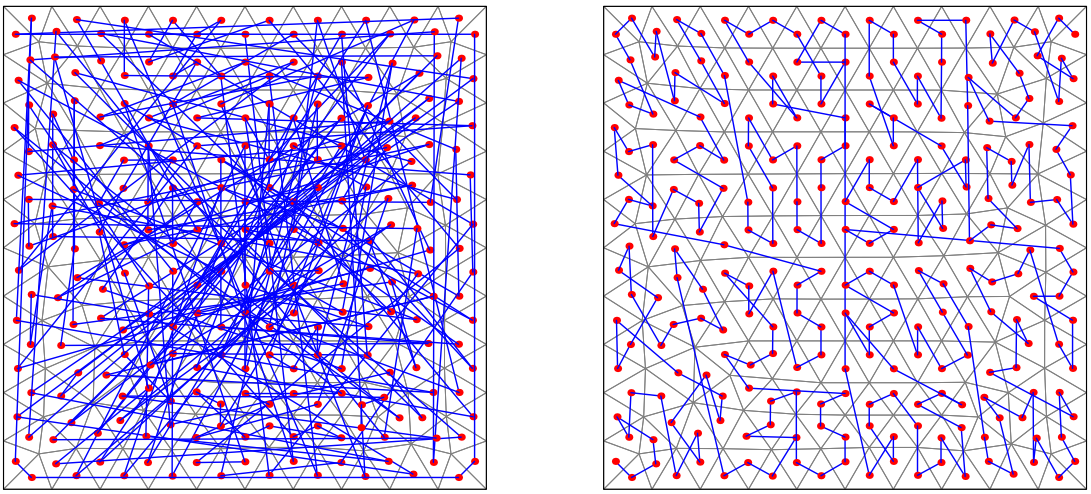
Efektivita algoritmů pracujících s nestrukturovanými daty závisí na očíslování jednotlivých prvků. Z tohoto pohledu nás v algoritmu 1 zajímá zejména struktura matice $\mathbf{A}$, která se odvíjí od očíslování

množiny nadstěn sítě $\mathcal{E}_h$. Nejvýznamnější operace s maticí $\mathbf{A}$ v algoritmu 1 je násobení matice a vektoru v rámci iterační metody pro řešení soustavy lineárních algebraických rovnic v kroku 6.7. Testovací úlohy porovnávající přímo efektivitu operací s daty nestrukturované sítě zde neuvádíme, ale budou předmětem dalšího výzkumu a odborných publikací.

Pro generování trojúhelníkové sítě pro úlohy popsané v kapitole 5 byl použit frontální algoritmus programu `Gmsh` [21], který vede k očíslování množiny elementů $\mathcal{K}_h = \{K_1, K_2, \ldots\}$ zobrazenému na obr. 4.2a pro síť $2D_1^{\triangle}$ (viz také tab. 5.1). Při číslování nadstěn $\mathcal{E}_h$ pak nejprve očíslujeme nadstěny $\mathcal{E}_{K_1}$, poté nadstěny $\mathcal{E}_{K_2}$, které ještě nebyly číslovány, atd. Struktura matice $\mathbf{A}$ pro $n = 2$ příslušná síti $2D_2^{\triangle}$ očíslované tímto způsobem je zobrazena na obr. 4.3a pomocí lomené čáry spojující středy po sobě jdoucích elementů. Zdůrazněme, že ačkoli to podle obrázku nevypadá, počet nenulových prvků na každém řádku je nejvýše 10. Nenulové prvky matice jsou však rozmístěny nesouvisle, což může v závislosti na velikosti matice a typu výpočetní architektury vést k nízké efektivitě algoritmů pracujících s touto maticí.

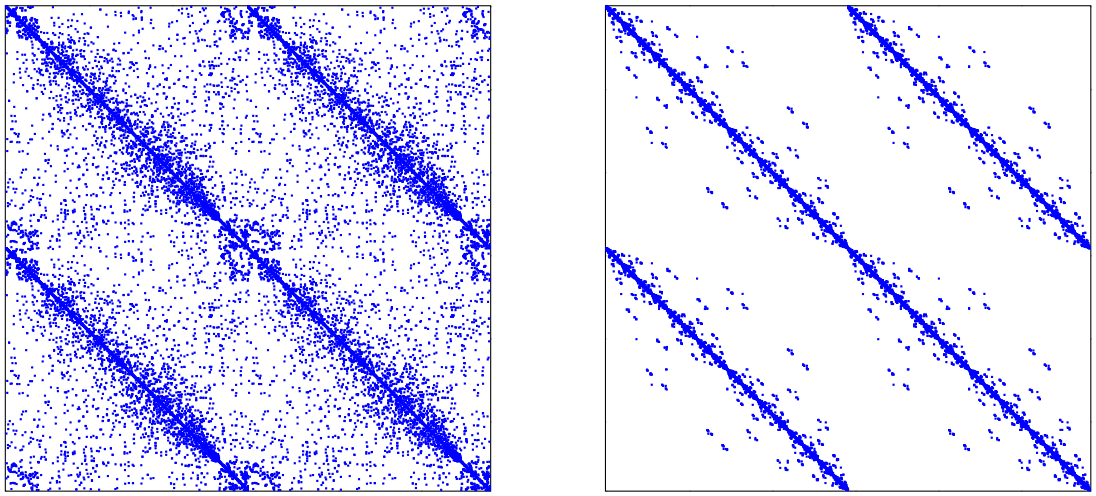
Pro zlepšení struktury matice $\mathbf{A}$ použijeme následující algoritmus pro očíslování entit sítě. Pro všechny entity dané dimenze najdeme jejich střed, z nichž vytvoříme d-rozměrný strom [4]. Dále tento strom projdeme strategií *in-order* a postupně budeme číslovat entity sítě příslušné danému uzlu stromu. Tímto postupem dostaneme očíslování zobrazené na obr. 4.2b pro síť $2D_1^{\triangle}$. Oproti předchozímu očíslování z obr. 4.2a je křivka znázorňující pořadí entit mnohem kratší a méně často se kříží, nové očíslování tedy lépe vystihuje blízkost entit v eukleidovské metrice. Také nenulové prvky matice $\mathbf{A}$ pro síť očíslovanou tímto způsobem jsou oproti předchozímu očíslování rozmístěny v souvislejších blocích podél diagonál, viz obr. 4.3b pro síť $2D_2^{\triangle}$.

Vliv očíslování entit sítě na efektivitu algoritmu 1 demonstrujeme na úloze popsané v sekci 5.4, kterou počítáme ve 2D na trojúhelníkové síti na procesoru Intel Core i7-5820K (6 jader, 3,3 GHz, 15 MiB cache) a GPU Nvidia Tesla K40 (2880 jader, 875 MHz). Výsledky uvedené v tab. 4.4 ukazují, že při výpočtech na CPU dochází pro jemné sítě k výraznému zpomalení při očíslování pomocí programu `Gmsh` oproti očíslování pomocí d-rozměrného stromu. Pro nejjemnější síť $2D_5^{\triangle}$ je rozdíl 17-28% v závislosti na počtu vláken a ortogonalizační metodě. Tento rozdíl je způsoben neefektivním využitím paměti cache při násobení matice a vektoru, které však při použití metody GMRES zabírá jen malou část celkové doby výpočtu. Při použití iteračních metod využívajících krátké rekurence (např. BiCGStab, TFQMR) by tak rozdíl mohl být daleko větší. Při výpočtu na GPU není rozdíl v rychlosti výpočtu tak výrazný jako na CPU, na nejjemnější síti $2D_5^{\triangle}$ je to 5-13%. Oproti CPU mají GPU mnohem menší množství paměti cache a efektivitu výpočtu zajišťují jinými způsoby jako např. slučováním přístupů do globální paměti pro sousední vlákna, což je však také ovlivněno uspořádáním nestrukturovaných dat.



(a) Frontální algoritmus programu `Gmsh` (b) 2-d strom

Obrázek 4.2: Očíslování nejhrubší trojúhelníkové sítě $2D_1^{\triangle}$ generovaná různými algoritmy znázorněné lomenou čárou spojující středy po sobě jdoucích elementů.



(a) Frontální algoritmus programu `Gmsh` (b) 2-d strom

Obrázek 4.3: Struktura matice $\mathbf{A}$ při různých očíslování sítě $2D_2^{\triangle}$.

		Intel Core i7-5820K											
		1 vlákno			2 vlákna			4 vlákna			6 vláken		
	Id.	`Gmsh`	strom	s/G	`Gmsh`	strom	s/G	`Gmsh`	strom	s/G	`Gmsh`	strom	s/G
MGSR	$2D_1^\triangle$	0,4	0,4	0,91	0,5	0,5	1,04	0,5	0,5	0,93	0,6	0,6	0,92
	$2D_2^\triangle$	5,1	5,0	0,99	4,0	3,9	0,97	3,3	3,1	0,96	3,6	3,6	1,00
	$2D_3^\triangle$	99,9	98,5	0,99	61,4	59,5	0,97	39,1	38,3	0,98	36,2	35,7	0,99
	$2D_4^\triangle$	2 662	2 383	0,90	1 454	1 299	0,89	787,3	701,0	0,89	632,5	573,6	0,91
	$2D_5^\triangle$	64 145	45 953	0,72	34 888	25 512	0,73	20 250	14 603	0,72	15 687	11 976	0,76
CWY	$2D_1^\triangle$	0,4	0,4	1,00	0,3	0,3	1,04	0,2	0,2	1,02	0,2	0,2	0,98
	$2D_2^\triangle$	6,3	6,2	0,99	3,8	3,7	0,99	2,5	2,3	0,94	2,2	2,0	0,93
	$2D_3^\triangle$	124,2	122,0	0,98	69,5	67,7	0,97	41,4	40,3	0,97	33,5	32,5	0,97
	$2D_4^\triangle$	2 965	2 696	0,91	1 646	1 481	0,90	941,4	855,2	0,91	732,3	671,7	0,92
	$2D_5^\triangle$	78 243	57 404	0,73	41 667	32 101	0,77	24 376	18 814	0,77	19 772	16 414	0,83

		Nvidia Tesla K40		
	Id.	`Gmsh`	strom	s/G
MGSR	$2D_1^\triangle$	4,4	4,4	1,00
	$2D_2^\triangle$	21,1	21,0	1,00
	$2D_3^\triangle$	114,3	114,0	1,00
	$2D_4^\triangle$	740,8	717,0	0,97
	$2D_5^\triangle$	7 198,3	6 829,2	0,95
CWY	$2D_1^\triangle$	1,3	1,3	1,00
	$2D_2^\triangle$	7,9	7,9	1,00
	$2D_3^\triangle$	47,0	46,2	0,98
	$2D_4^\triangle$	374,4	351,5	0,94
	$2D_5^\triangle$	3 913,2	3 387,0	0,87

Tabulka 4.4: Srovnání doby výpočtu úlohy ze sekce 5.4 na nestrukturovaných trojúhelníkových sítích při různých očíslováních: pomocí frontálního algoritmu programu `Gmsh` a pomocí 2-d stromu. Sloupec s/G označuje poměr dob výpočtu při očíslováních pomocí 2-d stromu a pomocí programu `Gmsh`.

Kapitola 5

Numerické simulace

V této kapitole se budeme zabývat numerickými simulacemi úloh nemísivého dvoufázového proudění a kompozičního dvoufázového proudění v porézním prostředí, které byly popsány v kapitole 1. V jednotlivých sekcích si nejprve připravíme nástroje pro numerickou analýzu pomocí experimentálního řádu konvergence a pro analýzu paralelizace a poté popíšeme testovací úlohy v homogenním prostředí se známým semi-analytickým řešením, které použijeme pro ověření konvergence numerického schématu popsaného v kapitole 3. Pro obě úlohy také porovnáme efektivitu paralelního výpočtu na strukturovaných a nestrukturovaných sítích. Část uvedených výsledků bude také publikována v článku [16], paralelní výpočty na nestrukturovaných sítích byly provedeny později.

V předchozí práci autora [28] byly pro verifikaci numerického schématu použity jednorozměrné difúzní úlohy pro nemísivé dvoufázové proudění v homogenním a heterogenním prostředí a jednorozměrné a dvojrozměrné advekčně-difúzní úlohy v heterogenním prostředí, jejichž výsledky lze srovnat s referenčními řešeními uvedenými v literatuře. V předchozí práci byly pro numerické řešení dvojrozměrných úloh použity pouze strukturované obdélníkové sítě. V této kapitole verifikujeme numerické schéma pomocí dvojrozměrných a trojrozměrných testovacích úloh dvoufázového a kompozičního proudění. Pro výpočty použijeme strukturované sítě tvořené čtverci nebo krychlemi a nestrukturované sítě tvořené trojúhelníky nebo čtyřstěny a pro všechny typy sítí porovnáme efektivitu paralelního výpočtu.

5.1 Nástroje pro numerickou analýzu

Pro danou síť $\mathcal{K}_h$ definujeme chybu aproximace $E_{h,g}$ skalární veličiny $g = g(\boldsymbol{x}, t)$ vztahem

$$E_{h,g} = g_{ex} - g_h, \tag{5.1}$$

kde g_{ex} je přesné řešení a g_h je numerická aproximace veličiny g. Pro analýzu úloh v následujících sekcích volíme za veličinu g buď saturaci nesmáčivé fáze S_n v případě dvoufázového proudění nebo hmotnostní zlomek X v případě kompozičního proudění.

Předpokládáme, že funkce g_{ex} a g_h jsou prvky prostoru $L^p(\Omega)$ pro $p = 1$ a 2. Pro analýzu řádu konvergence předpokládáme, že v čase $t = T$ má L^p norma chyby $E_{h,g}$ tvar

$$\|E_{h,g}\|_p = C_{T,g,p}\tau + C_{\Omega,g,p}h^{oc_{g,p}}, \tag{5.2}$$

kde h je parametr sítě $\mathcal{K}_h$, τ označuje konstantní časový krok pro danou síť, $C_{T,g,p}$ a $C_{\Omega,g,p}$ jsou nějaké kladné konstanty a $oc_{g,p}$ je řád konvergence numerického schématu v L^p normě vzhledem k veličině g. Řád konvergence $oc_{g,p}$ lze aproximovat pomocí tzv. experimentálního řádu konvergence $eoc_{g,p}$ definovaného vztahem

$$eoc_{g,p} = \frac{\ln\|E_{h_1,g}\|_p - \ln\|E_{h_2,g}\|_p}{\ln h_1 - \ln h_2}, \tag{5.3}$$

kde h_1 a h_2 jsou parametry dvou různých sítí $\mathcal{K}_{h_1}$, resp. $\mathcal{K}_{h_2}$.

Při výpočtu norem $\|E_{h,g}\|$ ve výrazu (5.3) nejprve rozložíme definiční integrál na součet integrálů přes elementy $K \in \mathcal{K}_h$:

$$\|E_{h,g}\| = \left(\int_\Omega |E_{h,g}|^p \right)^{\frac{1}{p}} = \left(\sum_{K \in \mathcal{K}_h} \int_K |E_{h,g}|^p \right)^{\frac{1}{p}}. \tag{5.4}$$

Integrály přes jednotlivé elementy sítě pak pomocí věty o substituci převedeme na integrály přes referenční element a pomocí Fubiniho věty přejdeme k jednorozměrným integrálům, které počítáme numericky pomocí Gaussovy–Kronrodovy kvadratury (G7–K15) s adaptivním dělením intervalu, která je implementována v C++11 knihovně `qdt` [36].

5.2 Nástroje pro analýzu paralelizace

Účelem paralelizace je zkrátit dobu výpočtu, kterou značíme symbolem CT. Přínos paralelizace algoritmu na vícejádrovém CPU můžeme vyjádřit například pomocí urychlení Sp, které je definováno jako podíl doby výpočtu při sériovém zpracování a doby výpočtu za pomoci ℓ vláken:

$$Sp = \frac{CT \text{ pro 1 vlákno}}{CT \text{ pro } \ell \text{ vláken}}. \tag{5.5}$$

Další důležitou veličinou je efektivita Eff, která vyjadřuje škálovatelnost výpočtu na vícejádrovém CPU pomocí ℓ vláken jako podíl urychlení a počtu vláken:

$$Eff = \frac{CT \text{ pro 1 vlákno}}{\ell \, (CT \text{ pro } \ell \text{ vláken})}. \tag{5.6}$$

Při výpočtu na GPU nemá smysl uvažovat sériový výpočet a kvůli zásadním hardwarovým rozdílům mezi architekturami GPU a CPU nemá smysl zavádět efektivitu pomocí počtu vláken GPU. Jediným nástrojem pro porovnání výpočtů na CPU a GPU tak je urychlení GSp definované jako podíl mezi výpočetními časy CT na CPU a GPU:

$$GSp = \frac{CT \text{ pro CPU}}{CT \text{ pro GPU}}. \tag{5.7}$$

5.3 Sítě pro numerické simulace

Pro numerické simulace úloh dvoufázového i kompozičního proudění použijeme stejné numerické sítě, jejichž parametry uvádíme ve společné tabulace 5.1. Ve všech případech se jedná o diskretizaci prostorové oblasti Ω tvořené jednotkovým čtvercem ve 2D a jednotkovou krychlí ve 3D. Geometrická konfigurace úloh, stejně jako volba časových kroků $\tau_{B\&C}$ a τ_{vG} uvedených v tab. 5.1, bude vysvětlena v následujících sekcích.

Strukturované sítě tvořené čtverci ($2D^{\square}$) a krychlemi ($3D^{\square}$) byly vytvořeny přímo pomocí knihovny `TNL`, nestrukturované trojúhelníkové sítě ($2D^{\triangle}$) byly vygenerovány pomocí programu `Gmsh` [21] a pro generování nestrukturovaných trojrozměrných sítí tvořených čtyřstěny ($3D^{\triangle}$) jsme použili program `COMSOL Multiphysics`. Sítě vygenerované programem `Gmsh` byly kvůli lepší efektivitě výpočtu přečíslovány pomocí postupu popsaného v sekci 4.4.2, očíslování sítí generovaných programem `COMSOL Multiphysics` je rovnou zřejmě optimální, tj. přečíslováním se ho již nepodařilo vylepšit. Poznamenejme, že není k dispozici dokumentace ke způsobu číslování sítě pro program `COMSOL Multiphysics`.

Id.	$\#\mathcal{K}_h$	h [m]	Dvoufázové proudění N_{dof}	$\tau_{B\&C}$ [s]	τ_{vG} [s]	Kompoziční proudění N_{dof}	$\tau_{B\&C}$ [s]	τ_{vG} [s]
$2D_1^{\square}$	225	$9{,}43 \cdot 10^{-2}$	960	253,16	317,00	1 440	253,16	317,00
$2D_2^{\square}$	900	$4{,}71 \cdot 10^{-2}$	3 720	90,50	80,00	5 580	90,50	80,00
$2D_3^{\square}$	3 600	$2{,}36 \cdot 10^{-2}$	14 640	31,90	19,96	21 960	31,90	19,96
$2D_4^{\square}$	14 400	$1{,}18 \cdot 10^{-2}$	58 080	10,62	5,02	87 120	10,62	5,02
$2D_5^{\square}$	57 600	$5{,}89 \cdot 10^{-3}$	231 360	3,57	1,26	347 040	3,57	1,26
$2D_1^{\triangle}$	242	$6{,}71 \cdot 10^{-2}$	766	454,55	36,36	1 149	454,55	36,36
$2D_2^{\triangle}$	944	$3{,}49 \cdot 10^{-2}$	2 912	145,99	15,49	4 368	145,99	15,49
$2D_3^{\triangle}$	3 714	$1{,}64 \cdot 10^{-2}$	11 302	44,64	4,64	16 953	44,64	4,64
$2D_4^{\triangle}$	14 788	$8{,}73 \cdot 10^{-3}$	44 684	13,44	1,35	67 026	13,44	1,35
$2D_5^{\triangle}$	59 336	$4{,}23 \cdot 10^{-3}$	178 648	5,00	0,50	267 972	5,00	0,50
$3D_1^{\square}$	3 375	$1{,}15 \cdot 10^{-1}$	21 600	333,33	235,29	32 400	200,00	200,00
$3D_2^{\square}$	27 000	$5{,}77 \cdot 10^{-2}$	167 400	131,58	58,82	251 100	12,50	12,50
$3D_3^{\square}$	216 000	$2{,}89 \cdot 10^{-2}$	1 317 600	53,48	14,71	1 976 400	0,78	0,78
$3D_4^{\square}$	1 728 000	$1{,}44 \cdot 10^{-2}$	10 454 400	22,10	3,68			
$3D_1^{\triangle}$	1 312	$2{,}13 \cdot 10^{-1}$	5 874	833,33	152,67	8 811	10,00	10,00
$3D_2^{\triangle}$	3 697	$1{,}27 \cdot 10^{-1}$	15 546	571,43	125,79	23 319	5,00	5,00
$3D_3^{\triangle}$	29 673	$6{,}29 \cdot 10^{-2}$	121 678	232,56	60,24	182 517	2,50	2,50
$3D_4^{\triangle}$	240 372	$3{,}48 \cdot 10^{-2}$	973 750	101,01	43,86	1 460 625	1,00	1,00
$3D_5^{\triangle}$	1 939 413	$1{,}84 \cdot 10^{-2}$	7 807 218	25,00	20,00			

Tabulka 5.1: Parametry sítí použitých pro numerické simulace: identifikátor sítě (Id.), počet elementů $\#\mathcal{K}_h$, parametr sítě h, počet stupňů volnosti N_{dof} a časové kroky $\tau_{B\&C}$ a τ_{vG} pro model Brookse a Coreyho, resp. van Genuchtena. Identifikátory označují sítě složené z čtverců ($2D^{\square}$), trojúhelníků ($2D^{\triangle}$), krychlí ($3D^{\square}$) nebo čtyřstěnů ($3D^{\triangle}$). Volba časových kroků pro jednotlivé úlohy bude vysvětlena později v příslušných sekcích.

5.4 Dvoufázové proudění

První testovací úlohou je úloha dvoufázového proudění popsaná v sekci 1.8. Koeficienty vyjadřující rovnice (1.16) pomocí obecné soustavy parciálních diferenciálních rovnic (2.1) jsou uvedeny v sekci 2.1. Pro speciální případy úlohy dvoufázového proudění v homogenním prostředí se zanedbanými vlivy gravitace je možné odvodit semi-analytická řešení [13, 15, 19, 33]. Nejobecnější postup v článku [15] popisuje úlohu s bodovým zdrojem v prostoru obecné dimenze $d \in \mathbb{N}$ a vede k radiálně symetrickému řešení. V této sekci se budeme zabývat úlohami pro $d = 2$ a $d = 3$.

Geometrická konfigurace úlohy ve 2D a 3D je zobrazena na obr. 5.1. Díky radiální symetrii přesného řešení [15] můžeme uvažovat pouze jeden kvadrant, resp. oktant celé oblasti ve 2D, resp. 3D. Pro oba případy uvažujeme stejné prostředí označené jako Písek A v tab. B.1 a stejné parametry tekutin voda a DNAPL uvedené v tab. B.2. V čase $t = 0$ je v celé oblasti konstantní saturace vody $S_w^{ini} = 0{,}95$ a bodovým zdrojem umístěným v počátku souřadného systému $\boldsymbol{x} = \mathbf{0}$ začneme do oblasti vtláčet nesmáčivou fázi s předepsaným objemovým tokem $Q_0[\mathrm{m^d s^{-1}}]$, jehož hodnota je dle [15] dána vztahem

$$Q_0 = Q_0(t) = A_d t^{\frac{d-2}{2}}, \tag{5.8}$$

kde $A_d[\mathrm{m^d s^{-\frac{d}{2}}}]$ je volitelný parametr toku Q_0. Hodnoty A_d, konečný čas simulace T a rozměry oblasti byly zvoleny tak, aby pro model Brookse a Coreyho i van Genuchtenův model vtláčená fáze nedosáhla

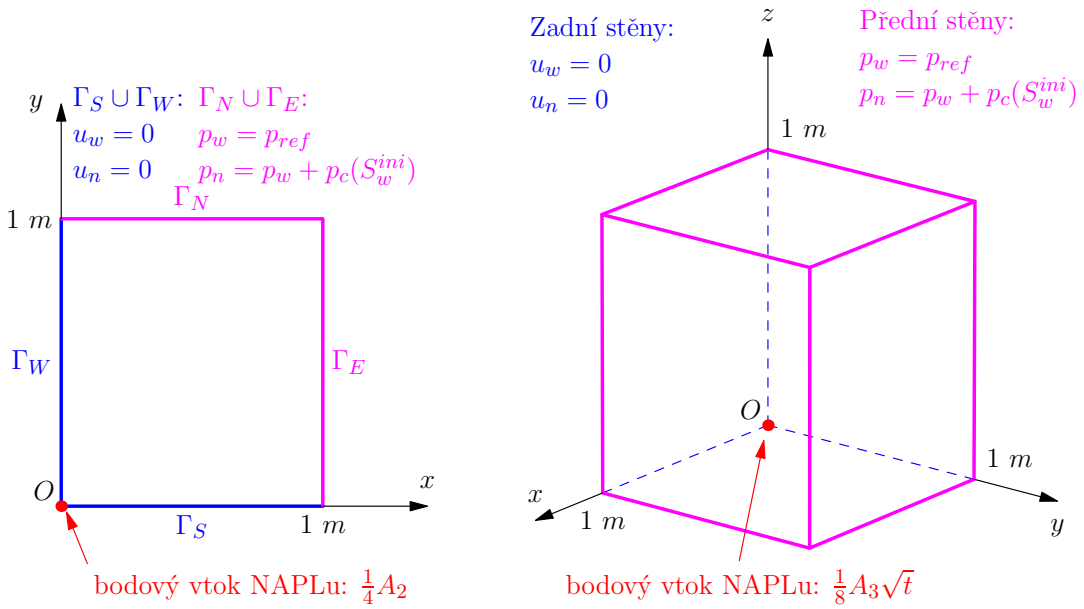
okolí části hranice aproximující hranici v nekonečnu. Zvolené rozměry oblastí jsou zobrazeny na obr. 5.1, konečný čas simulace volíme $T = 20\,000\,\mathrm{s}$ a hodnoty A_d volíme následovně: $A_2 = 10^{-5}\,\mathrm{m}^2\mathrm{s}^{-1}$, $A_3 = 10^{-7}\,\mathrm{m}^3\mathrm{s}^{-\frac{3}{2}}$.

Jak plyne z odvození numerického schématu v kapitole 3, okrajovou podmínku s bodovým zdrojem nelze přímo vyjádřit pomocí numerického schématu, ale je potřeba ji aproximovat pomocí nadstěn sítě $\mathcal{E}_h^{ext}$, které mají nějaký vrchol v počátku souřadného systému $\boldsymbol{x} = \mathbf{0}$. Tato situace je zobrazena na obr. 5.2, kde vtokovou část hranice $\Gamma = \partial\Omega$ značíme jako Γ_{in}. Na Γ_{in} předepisujeme nulovou Neumannovu podmínku pro rychlost smáčivé fáze $\boldsymbol{v}_w$ a nenulovou Neumannovu podmínku pro rychlost nesmáčivé fáze $\boldsymbol{v}_n$ tak, že

$$\int_{\Gamma_{in}} \boldsymbol{v}_n \cdot \boldsymbol{n} = -Q_0(t), \qquad \forall t \in [0, T]. \tag{5.9}$$

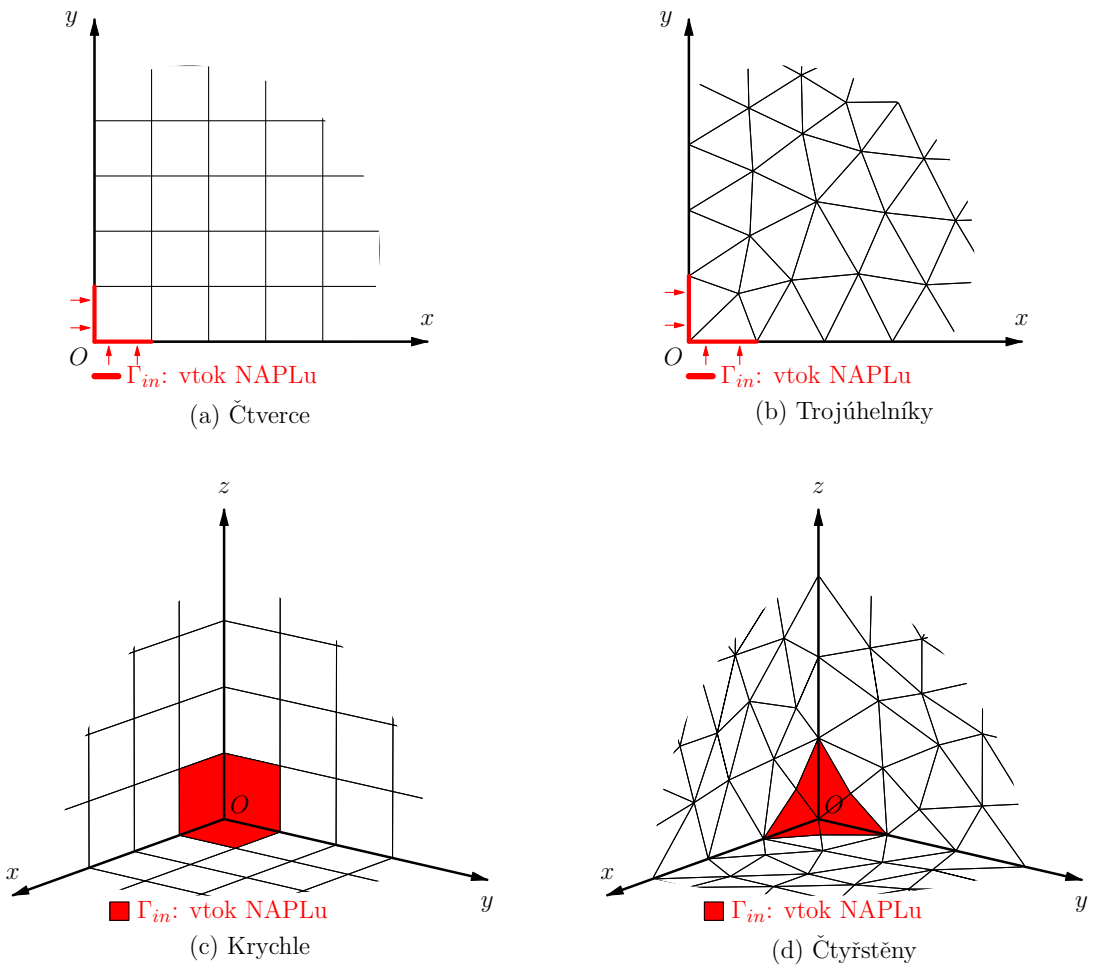
Zbývající okrajové podmínky z obr. 5.1 zůstávají nezměněny.

Pro řešení úlohy použijeme strukturované a nestrukturované sítě, jejichž parametry jsou uvedeny v tab. 5.1. Numerické řešení saturace S_n ve finálním čase $T = 20\,000\,\mathrm{s}$ pro několik sítí ve 2D je zobrazeno na obr. 5.3. V následující podsekci provedeme numerickou analýzu pomocí experimentálního řádu konvergence pro srovnání numerického a semi-analytického řešení.



(a) 2D (b) 3D

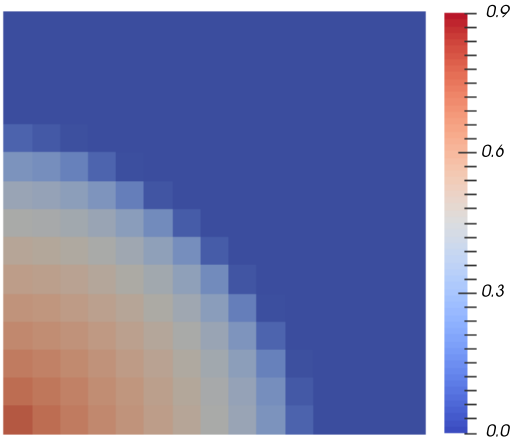
Obrázek 5.1: Geometrická konfigurace výpočetní oblasti včetně okrajových podmínek pro (a) dvojrozměrnou a (b) trojrozměrnou úlohu. Obrázek je převzat z [16].



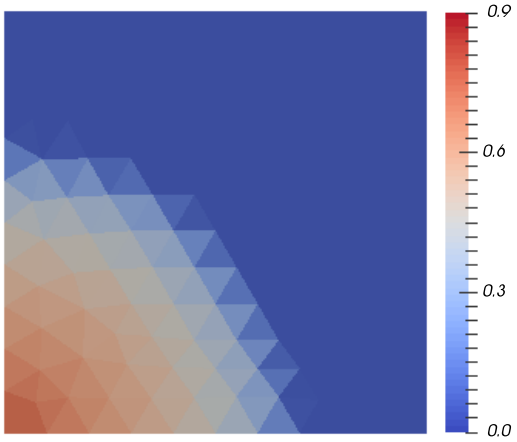
(a) Čtverce (b) Trojúhelníky

(c) Krychle (d) Čtyřstěny

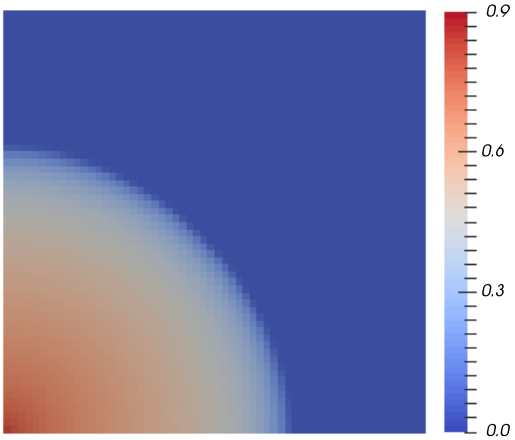
Obrázek 5.2: Aproximace okrajové podmínky pro bodový zdroj na sítích tvořených (a) čtverci, (b) trojúhelníky, (c) krychlemi a (d) čtyřstěny. Obrázek je převzat z [16].



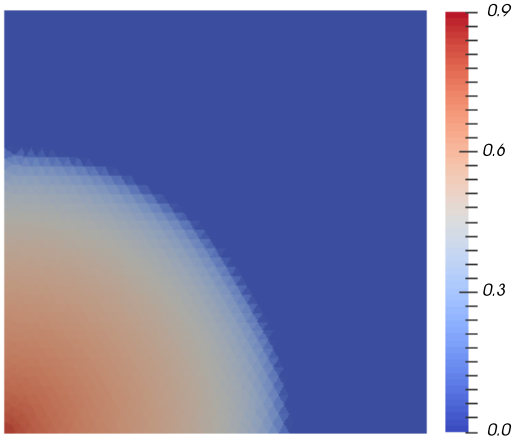
(a) $2D_1^{\square}$



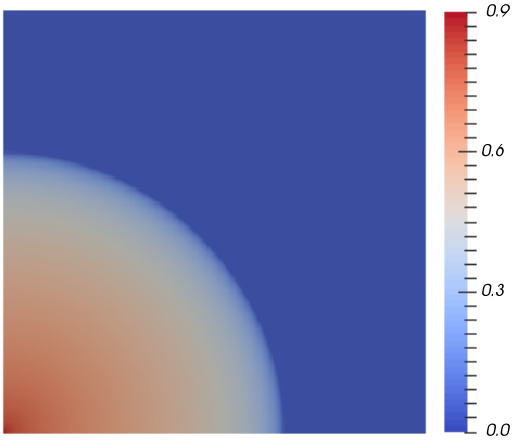
(b) $2D_1^{\triangle}$



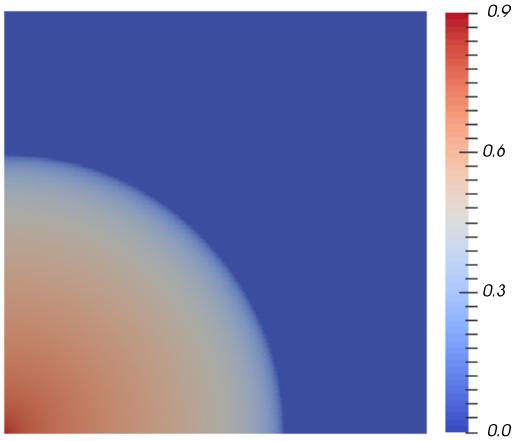
(c) $2D_3^{\square}$



(d) $2D_3^{\triangle}$



(e) $2D_5^{\square}$



(f) $2D_5^{\triangle}$

Obrázek 5.3: Numerické řešení saturace S_n pro úlohu dvoufázového proudění v čase $T = 20\,000$ s.

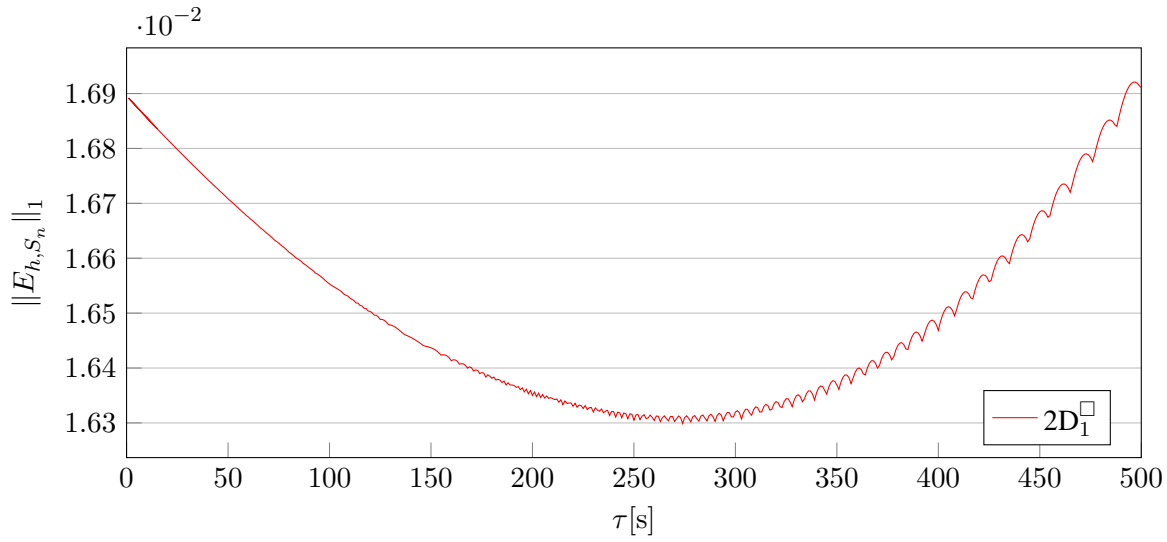
5.4.1 Numerická analýza

Numerickou analýzu provedeme pro saturaci nesmáčivé fáze S_n, pro kterou je pro danou úlohu známé semi-analytické řešení [15]. Při numerické analýze pomocí experimentálního řádu konvergence $eoc_{S_n,p}$ dle vztahu (5.3) je třeba při zjemňování sítě zmenšovat také časový krok, aby chyba časové diskretizace nepřevažovala nad chybou prostorové diskretizace. Zkoumaná úloha je ale komplikovaná tím, že chyba numerického řešení na dané síti $\mathcal{K}_h$ nezávisí monotónně na zvoleném časovém kroku τ. Pozorovaná závislost L^1 normy chyby numerického řešení pro nejhrubší síť $2D_1^{\square}$ a model Brookse a Coreyho je zobrazena na obr. 5.4, kde je chyba vyčíslena pro časové kroky $\tau \in \mathbb{N}$ a časový krok zůstává konstantní po celou dobu simulace až na poslední časový krok, který je volen tak, aby simulace skončila přesně v čase T. Na obr. 5.4 však pozorujeme, že lokální minima funkce $\|E_{h,S_n}\|_1$ odpovídají těm časovým krokům τ, pro které je finální čas T celočíselným násobkem τ. Podobné chování lze pozorovat i na ostatních sítích, proto dále vždy volíme časové kroky tak, aby byly konstantní po celou dobu simulace včetně posledního kroku.

Globální chování chyby pro model Brookse a Coreyho a van Genuchtenův model je pro několik sítí zobrazeno na obr. 5.5. Ve všech případech se při zjemnění sítě zachová globální trend, pouze se změní hodnoty chyby a dojde k posunu minim. V případě modelu Brookse a Coreyho lze snadno identifikovat globální minima funkcí $\|E_{h,S_n}\|_1$ pro každou síť a ty použít pro výpočet experimentálního řádu konvergence. Jelikož chybu pro danou síť bereme nejmenší možnou, hodnoty $eoc_{S_n,p}$ by při zjemňování sítě neměly oscilovat. Pro ušetření výpočetní kapacity pracoviště KM FJFI provádíme minimalizaci algoritmicky jen na několika nejhrubších sítích, přičemž si pomáháme i přechodnými sítěmi, které nejsou uvedeny v tab. 5.1. Pro jemnější sítě potom extrapolujeme očekávanou polohu minima dle výsledků na hrubých sítích. Pro extrapolaci předpokládáme závislost polohy minima τ_{min} na parametru sítě h dle vztahu

$$\tau_{min}(h) = Ch^{\gamma}, \tag{5.10}$$

kde parametry C a γ určujeme z výsledků na hrubých sítích pomocí regresní analýzy metodou nejmenších čtverců. Po přechodu na jemnější síť dále provedeme několik iterací minimalizačního algoritmu pro



Obrázek 5.4: Ilustrace závislost chyby numerického řešení na volbě časového kroku při použití modelu Brookse a Coreyho a sítě $2D_1^{\square}$.

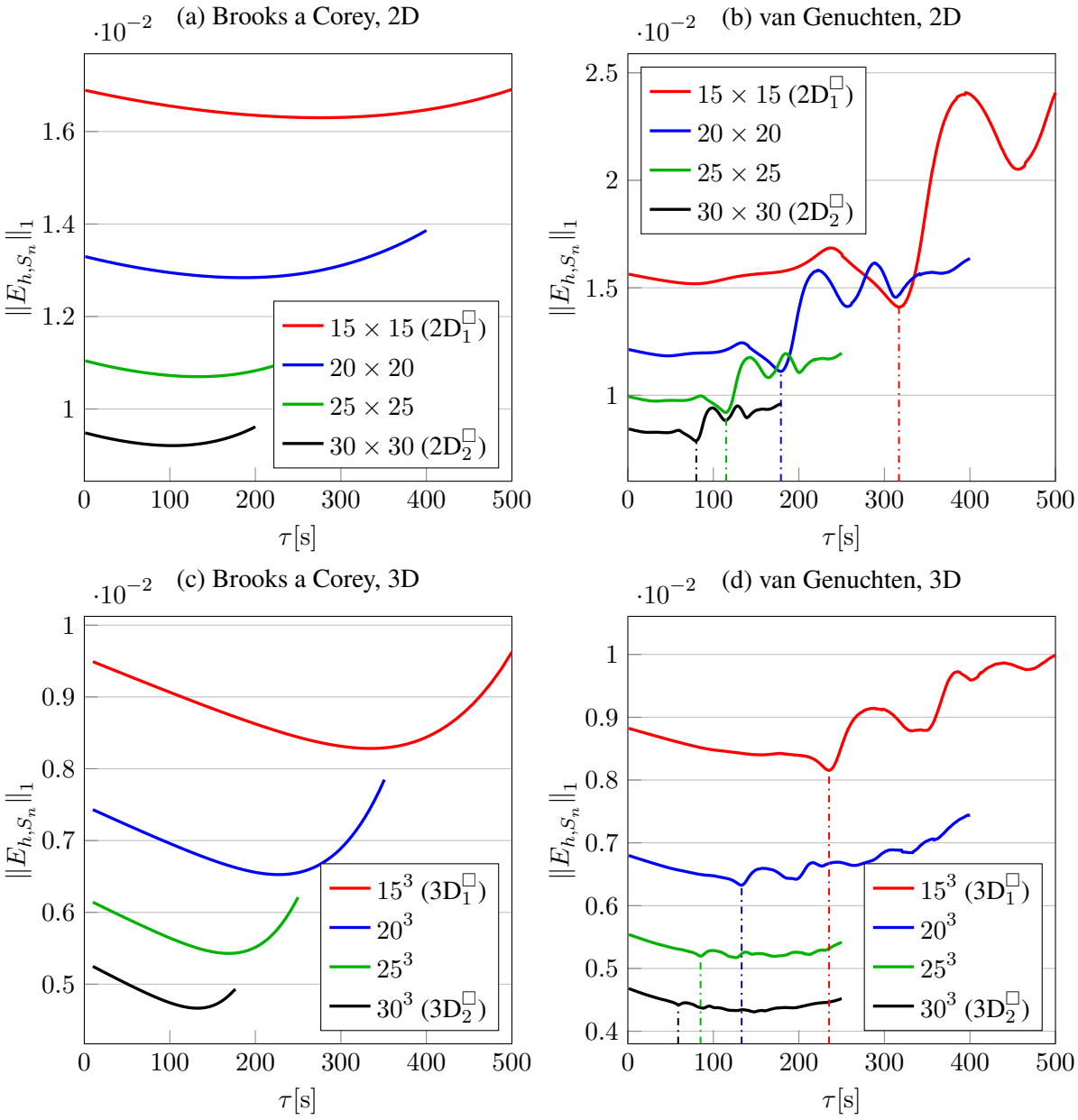
zpřesnění polohy minima a zlepšení extrapolace pro další síť. Výsledky regresní analýzy parametrů C a γ pro hodnoty h a τ zvolené dle tab. 5.1 jsou uvedeny v tab. 5.3 a vizualizovány na obr. 5.6 a 5.7.

V případě modelu van Genuchtena je průběh chyby komplikovanější a pro velké časové kroky dochází k výrazným oscilacím. Navíc, jak je vidět na obr. 5.5d pro strukturované trojrozměrné sítě, poloha globálního minima při zjemňování sítě nemusí odpovídat kratšímu časovému kroku oproti předchozí síti. Dále pozorujeme, že krajní lokální minima pozorovatelná na nejhrubší síti mohou při zjemňování sítě postupně zmizet. Proto v tomto případě nehledáme globální minima, ale nejprve pro několik nejhrubších sítí určíme polohy výrazných lokálních minim pro co nejkratší časový krok (na obr. 5.5 vyznačeny svislou přerušovanou čarou), z nichž extrapolujeme parametry C a γ pro závislost (5.10), v které pokračujeme pro jemnější sítě nehledě na to, zda se jedná o lokální minimum.

Konkrétní volba časových kroků při výpočtu experimentálních řádů konvergence pro jednotlivé sítě je uvedena v tab. 5.1. Výsledky numerické analýzy uvedené v tab. 5.2 naznačují, že numerické schéma konverguje ve všech případech s prvním řádem přesnosti. Experimentální řád konvergence v L^2 normě vychází mírně menší oproti řádu konvergence v L^1 normě, vůči které jsme prováděli minimalizaci. Na strukturovaných sítích se hodnoty $eoc_{S_n,p}$ mezi sebou příliš neliší, na nestrukturovaných sítích jsou rozdíly výraznější, ale při zjemňování sítě se rozdíly zmenšují.

	Brooks & Corey				van Genuchten			
Id.	$\|E_{h,S_n}\|_1$	$eoc_{S_n,1}$	$\|E_{h,S_n}\|_2$	$eoc_{S_n,2}$	$\|E_{h,S_n}\|_1$	$eoc_{S_n,1}$	$\|E_{h,S_n}\|_2$	$eoc_{S_n,2}$
$2D_1^{\square}$	$1{,}52 \cdot 10^{-2}$		$3{,}26 \cdot 10^{-2}$		$1{,}41 \cdot 10^{-2}$		$2{,}17 \cdot 10^{-2}$	
		0,80		**0,65**		**0,84**		**0,81**
$2D_2^{\square}$	$8{,}75 \cdot 10^{-3}$		$2{,}08 \cdot 10^{-2}$		$7{,}88 \cdot 10^{-3}$		$1{,}24 \cdot 10^{-2}$	
		0,82		**0,62**		**0,87**		**0,86**
$2D_3^{\square}$	$4{,}97 \cdot 10^{-3}$		$1{,}35 \cdot 10^{-2}$		$4{,}31 \cdot 10^{-3}$		$6{,}83 \cdot 10^{-3}$	
		0,85		**0,60**		**0,88**		**0,88**
$2D_4^{\square}$	$2{,}76 \cdot 10^{-3}$		$8{,}93 \cdot 10^{-3}$		$2{,}34 \cdot 10^{-3}$		$3{,}72 \cdot 10^{-3}$	
		0,87		**0,63**		**0,86**		**0,85**
$2D_5^{\square}$	$1{,}51 \cdot 10^{-3}$		$5{,}79 \cdot 10^{-3}$		$1{,}29 \cdot 10^{-3}$		$2{,}06 \cdot 10^{-3}$	
$2D_1^{\triangle}$	$1{,}54 \cdot 10^{-2}$		$3{,}25 \cdot 10^{-2}$		$1{,}43 \cdot 10^{-2}$		$2{,}13 \cdot 10^{-2}$	
		0,97		**0,84**		**0,97**		**0,93**
$2D_2^{\triangle}$	$8{,}14 \cdot 10^{-3}$		$1{,}89 \cdot 10^{-2}$		$7{,}58 \cdot 10^{-3}$		$1{,}16 \cdot 10^{-2}$	
		0,80		**0,61**		**0,84**		**0,83**
$2D_3^{\triangle}$	$4{,}44 \cdot 10^{-3}$		$1{,}19 \cdot 10^{-2}$		$4{,}01 \cdot 10^{-3}$		$6{,}22 \cdot 10^{-3}$	
		0,96		**0,67**		**1,01**		**1,00**
$2D_4^{\triangle}$	$2{,}41 \cdot 10^{-3}$		$7{,}79 \cdot 10^{-3}$		$2{,}12 \cdot 10^{-3}$		$3{,}30 \cdot 10^{-3}$	
		0,86		**0,64**		**0,85**		**0,84**
$2D_5^{\triangle}$	$1{,}29 \cdot 10^{-3}$		$4{,}90 \cdot 10^{-3}$		$1{,}15 \cdot 10^{-3}$		$1{,}79 \cdot 10^{-3}$	
$3D_1^{\square}$	$8{,}28 \cdot 10^{-3}$		$2{,}59 \cdot 10^{-2}$		$8{,}15 \cdot 10^{-3}$		$1{,}64 \cdot 10^{-2}$	
		0,83		**0,70**		**0,88**		**0,86**
$3D_2^{\square}$	$4{,}67 \cdot 10^{-3}$		$1{,}59 \cdot 10^{-2}$		$4{,}42 \cdot 10^{-3}$		$9{,}06 \cdot 10^{-3}$	
		0,84		**0,69**		**0,90**		**0,89**
$3D_3^{\square}$	$2{,}60 \cdot 10^{-3}$		$9{,}87 \cdot 10^{-3}$		$2{,}36 \cdot 10^{-3}$		$4{,}90 \cdot 10^{-3}$	
		0,86		**0,69**		**0,93**		**0,92**
$3D_4^{\square}$	$1{,}44 \cdot 10^{-3}$		$6{,}12 \cdot 10^{-3}$		$1{,}24 \cdot 10^{-3}$		$2{,}58 \cdot 10^{-3}$	
$3D_1^{\triangle}$	$1{,}15 \cdot 10^{-2}$		$3{,}48 \cdot 10^{-2}$		$1{,}21 \cdot 10^{-2}$		$2{,}43 \cdot 10^{-2}$	
		0,69		**0,62**		**0,77**		**0,73**
$3D_2^{\triangle}$	$8{,}02 \cdot 10^{-3}$		$2{,}52 \cdot 10^{-2}$		$8{,}13 \cdot 10^{-3}$		$1{,}66 \cdot 10^{-2}$	
		0,86		**0,75**		**0,93**		**0,90**
$3D_3^{\triangle}$	$4{,}41 \cdot 10^{-3}$		$1{,}49 \cdot 10^{-2}$		$4{,}26 \cdot 10^{-3}$		$8{,}83 \cdot 10^{-3}$	
		1,02		**0,93**		**1,14**		**1,13**
$3D_4^{\triangle}$	$2{,}40 \cdot 10^{-3}$		$8{,}62 \cdot 10^{-3}$		$2{,}16 \cdot 10^{-3}$		$4{,}53 \cdot 10^{-3}$	
		1,01		**0,71**		**1,08**		**1,08**
$3D_5^{\triangle}$	$1{,}26 \cdot 10^{-3}$		$5{,}48 \cdot 10^{-3}$		$1{,}09 \cdot 10^{-3}$		$2{,}28 \cdot 10^{-3}$	

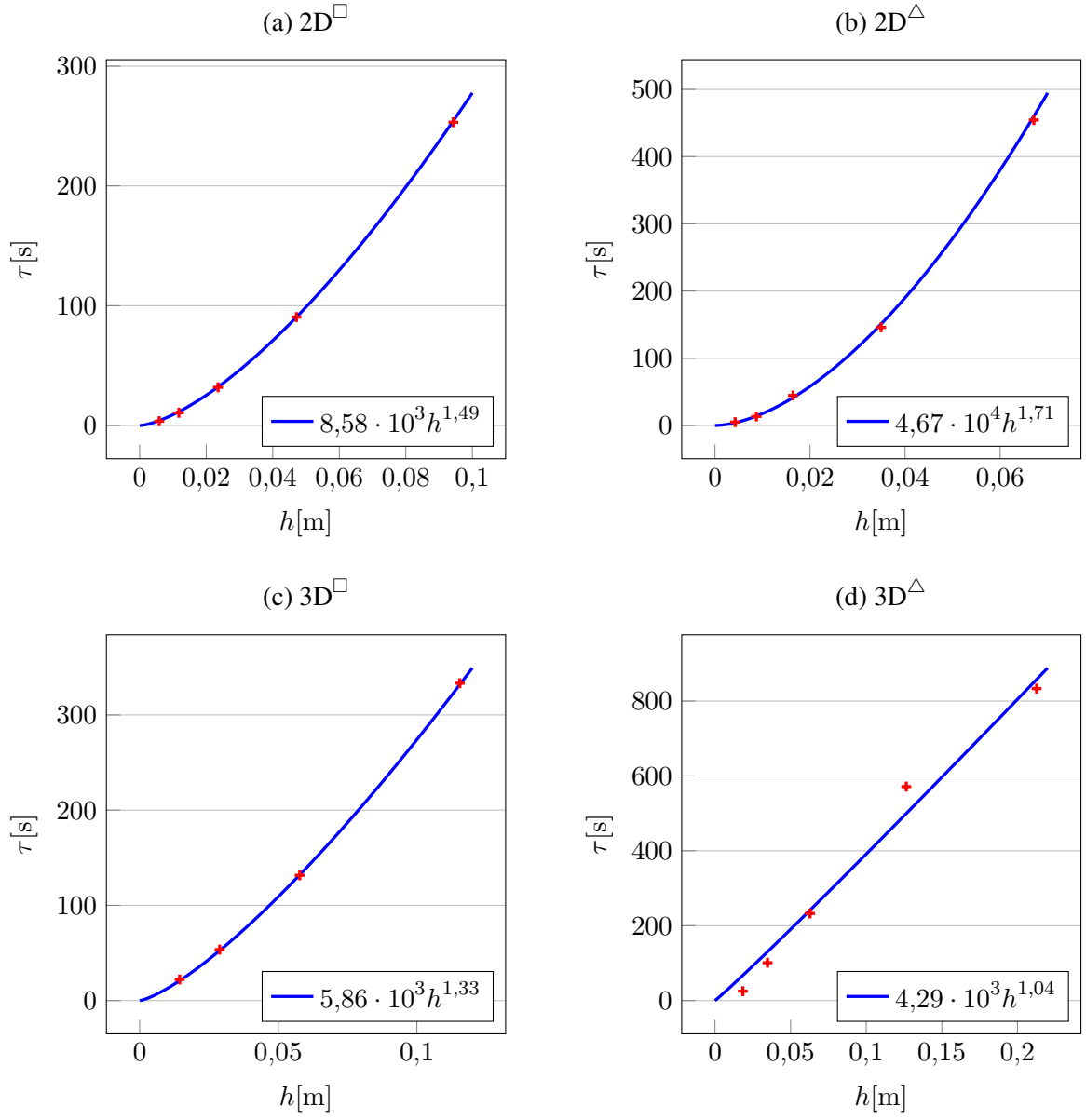
Tabulka 5.2: Výsledky numerické analýzy pomocí L^1 a L^2 norem chyby E_{h,S_n}.



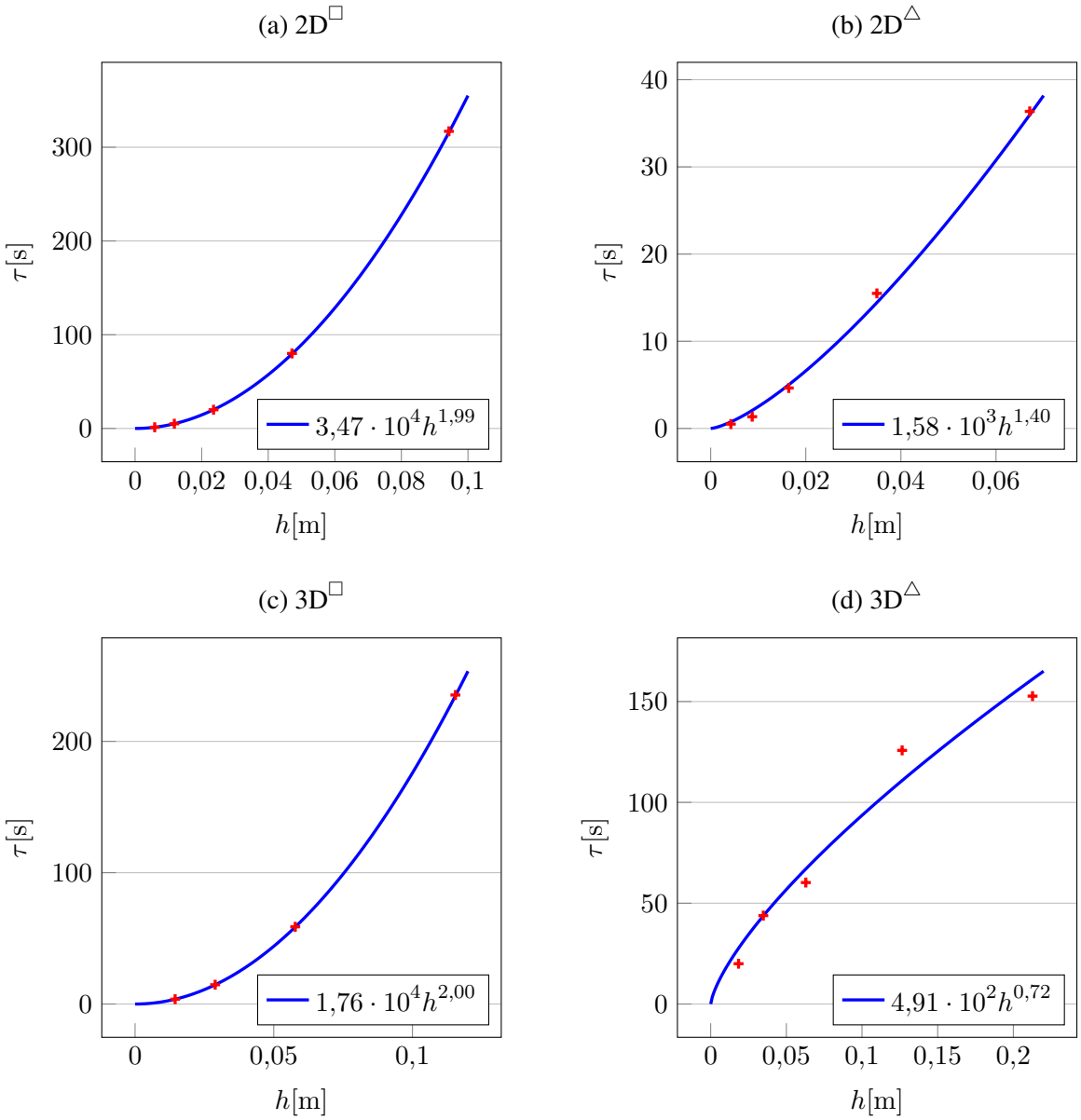
Obrázek 5.5: Závislosti chyby numerického řešení na časovém kroku pro model Brookse a Coreyho a van Genuchtenův model na strukturovaných sítích ve 2D a 3D. Časový krok je volen tak, aby byl konstantní po celou dobu simulace včetně posledního kroku a pro účely vizualizace je mezi těmito hodnotami v grafech provedena lineární interpolace. V případě modelu van Genuchtena vyznačuje přerušovaná čára polohu vybraného lokálního minima pro každou síť.

Id.	Brooks a Corey C	γ	van Genuchten C	γ
2D$^{\square}$	$8{,}58 \cdot 10^3$	1,49	$3{,}47 \cdot 10^4$	1,99
2D$^{\triangle}$	$4{,}67 \cdot 10^4$	1,71	$1{,}58 \cdot 10^3$	1,40
3D$^{\square}$	$5{,}86 \cdot 10^3$	1,33	$1{,}76 \cdot 10^4$	2,00
3D$^{\triangle}$	$4{,}29 \cdot 10^3$	1,04	$4{,}91 \cdot 10^2$	0,72

Tabulka 5.3: Výsledky regresní analýzy parametrů C a γ ve vztahu (5.10) pro hodnoty h a τ uvedené v tab. 5.1.



Obrázek 5.6: Regresní křivky (5.10) a hodnoty h a τ uvedené v tab. 5.1 pro model Brookse a Coreyho.



Obrázek 5.7: Regresní křivky (5.10) a hodnoty h a τ uvedené v tab. 5.1 pro model van Genuchtena.

5.4.2 Efektivita paralelního výpočtu

Pro úlohu dvoufázového proudění popsanou v sekci 5.4 dále provedeme analýzu efektivity výpočtu na CPU s využitím různého počtu jader a GPU. V tab. 5.4 a 5.5 porovnáváme dobu výpočtu CT, efektivitu Eff paralelního výpočtu na CPU a urychlení GSp výpočtu na GPU pro úlohu popsanou výše s nastavením odpovídajícím výsledkům numerické analýzy v tab. 5.2 pro model Brookse a Coreyho. Výpočty probíhaly na GPU Nvidia Tesla K40 (2880 jader, 875 MHz, 12 GB globální paměti typu GDDR5) a CPU Intel Core i7-5820K (6 jader, 3,3 GHz) v systému s 64 GB operační paměti typu DDR3. Pro zajištění relevantních hodnot efektivity Eff byla při všech výpočtech na CPU byla deaktivována technologie Intel Turbo Boost, která by jinak nedeterministicky zkreslila výsledky analýzy.

Při výpočtech používáme všechny techniky pro zrychlení výpočtu popsané v kapitole 4. Výsledky v tab. 5.4 a 5.5 ukazují, že pro dostatečně jemné sítě je výpočet na GPU několikanásobně rychlejší oproti odpovídajícímu výpočtu na CPU. Zatímco na GPU je metoda GMRES s ortogonalizací pomocí kom-

		GPU	CPU									
			1 vlákno		2 vlákna			4 vlákna			6 vláken	
	Id.	CT	CT	GSp	CT	Eff	GSp	CT	Eff	GSp	CT	Eff	GSp
MGSR	$2D_1^{\square}$	5,1	0,6	0,12	0,7	0,45	0,13	0,8	0,19	0,15	0,9	0,11	0,17
	$2D_2^{\square}$	28,1	11,5	0,41	7,9	0,72	0,28	6,4	0,45	0,23	6,8	0,28	0,24
	$2D_3^{\square}$	117,1	173,6	1,48	95,9	0,91	0,82	61,2	0,71	0,52	52,8	0,55	0,45
	$2D_4^{\square}$	740,4	4 023,5	5,43	2 154,1	0,93	2,91	1 192,1	0,84	1,61	941,6	0,71	1,27
	$2D_5^{\square}$	8 237,3	82 323,5	9,99	47 982,0	0,86	5,82	26 919,0	0,76	3,27	19 915,5	0,69	2,42
CWY	$2D_1^{\square}$	1,5	0,7	0,45	0,4	0,79	0,28	0,3	0,52	0,22	0,3	0,41	0,18
	$2D_2^{\square}$	11,0	13,2	1,20	7,6	0,87	0,69	4,8	0,68	0,44	4,0	0,55	0,37
	$2D_3^{\square}$	46,3	197,0	4,25	107,5	0,92	2,32	65,7	0,75	1,42	52,6	0,62	1,14
	$2D_4^{\square}$	380,0	4 325,7	11,38	2 360,6	0,92	6,21	1 448,1	0,75	3,81	1 195,8	0,60	3,15
	$2D_5^{\square}$	4 449,9	91 166,3	20,49	49 004,3	0,93	11,01	29 182,1	0,78	6,56	24 684,0	0,62	5,55

(a) obdélníky

		GPU	CPU									
			1 vlákno		2 vlákna			4 vlákna			6 vláken	
	Id.	CT	CT	GSp	CT	Eff	GSp	CT	Eff	GSp	CT	Eff	GSp
MGSR	$2D_1^{\triangle}$	4,7	0,3	0,07	0,5	0,33	0,11	0,5	0,18	0,10	0,6	0,09	0,13
	$2D_2^{\triangle}$	22,4	5,0	0,22	3,9	0,65	0,17	3,1	0,40	0,14	3,6	0,23	0,16
	$2D_3^{\triangle}$	120,0	98,5	0,82	59,5	0,83	0,50	38,3	0,64	0,32	35,7	0,46	0,30
	$2D_4^{\triangle}$	778,3	2 382,8	3,06	1 298,8	0,92	1,67	701,0	0,85	0,90	573,5	0,69	0,74
	$2D_5^{\triangle}$	7 387,9	45 953,4	6,22	25 512,4	0,90	3,45	14 602,7	0,79	1,98	11 976,4	0,64	1,62
CWY	$2D_1^{\triangle}$	1,5	0,4	0,27	0,3	0,60	0,22	0,2	0,45	0,15	0,2	0,32	0,14
	$2D_2^{\triangle}$	8,9	6,2	0,70	3,7	0,84	0,42	2,3	0,66	0,26	2,0	0,52	0,23
	$2D_3^{\triangle}$	51,1	122,0	2,39	67,7	0,90	1,32	40,3	0,76	0,79	32,5	0,63	0,64
	$2D_4^{\triangle}$	396,1	2 695,6	6,80	1 480,7	0,91	3,74	855,2	0,79	2,16	671,7	0,67	1,70
	$2D_5^{\triangle}$	4 008,3	57 404,2	14,32	32 100,5	0,89	8,01	18 814,1	0,76	4,69	16 414,0	0,58	4,09

(b) trojúhelníky

Tabulka 5.4: Srovnání doby výpočtu CT, efektivity Eff paralelního výpočtu na CPU a urychlení GSp výpočtu na GPU při výpočtu úlohy popsané v sekci 5.4 ve 2D.

paktní WY reprezentace (CWY) vždy podstatně rychlejší oproti modifikovanému Gramovu–Schmidtovu procesu s reortogonalizací (MGSR), na CPU je metoda CWY většinou mírně pomalejší než metoda MGSR. Ve srovnání výsledků pro strukturované a nestrukturované sítě je vidět, že urychlení GSp je na nestrukturovaných sítích vždy o něco menší než na odpovídající strukturované síti. To je však způsobeno zejména tím, že urychlení závisí primárně na počtu stupňů volnosti N_{dof} dané sítě a nestrukturované sítě uvedené v tab. 5.1 mají menší počet stupňů volnosti oproti odpovídajícím strukturovaným sítím. Celkově můžeme konstatovat, že numerické schéma popsané v kapitole 3 lze dobře paralelizovat pro strukturované i nestrukturované sítě.

		GPU	CPU									
			1 vlákno		2 vlákna			4 vlákna			6 vláken	
	Id.	CT	CT	GSp	CT	Eff	GSp	CT	Eff	GSp	CT	Eff	GSp
MGSR	$3D_1^{\square}$	5,9	13,8	2,34	7,2	0,96	1,22	4,3	0,80	0,73	3,4	0,67	0,58
	$3D_2^{\square}$	55,7	524,6	9,42	304,7	0,86	5,47	173,7	0,76	3,12	128,2	0,68	2,30
	$3D_3^{\square}$	1 234,3	21 128,7	17,12	12 770,7	0,83	10,35	7 317,4	0,72	5,93	6 241,6	0,56	5,06
	$3D_4^{\square}$	44 798,3	(nepočítáno na 1, 2 a 4 vláknech)								272 104,0		6,07
CWY	$3D_1^{\square}$	2,1	15,2	7,30	8,0	0,96	3,82	4,4	0,86	2,13	3,4	0,75	1,62
	$3D_2^{\square}$	30,8	564,3	18,33	319,5	0,88	10,38	186,7	0,76	6,07	150,3	0,63	4,88
	$3D_3^{\square}$	828,0	20 569,5	24,84	12 406,1	0,83	14,98	7 092,6	0,73	8,57	5 533,7	0,62	6,68
	$3D_4^{\square}$	31 805,6	(nepočítáno na 1, 2 a 4 vláknech)								234 066,0		7,36

(a) kvádry

		GPU	CPU									
			1 vlákno		2 vlákna			4 vlákna			6 vláken	
	Id.	CT	CT	GSp	CT	Eff	GSp	CT	Eff	GSp	CT	Eff	GSp
MGSR	$3D_1^{\triangle}$	3,8	1,7	0,44	1,2	0,71	0,31	0,8	0,53	0,21	0,8	0,33	0,22
	$3D_2^{\triangle}$	6,1	7,2	1,19	4,3	0,84	0,70	2,6	0,70	0,43	2,3	0,53	0,37
	$3D_3^{\triangle}$	45,3	274,5	6,06	152,6	0,90	3,37	87,5	0,78	1,93	72,4	0,63	1,60
	$3D_4^{\triangle}$	873,1	11 270,0	12,91	6 228,3	0,90	7,13	3 414,9	0,83	3,91	3 187,9	0,59	3,65
	$3D_5^{\triangle}$	55 880,2	(nepočítáno na CPU)										
CWY	$3D_1^{\triangle}$	1,4	2,0	1,48	1,2	0,85	0,88	0,7	0,68	0,54	0,6	0,54	0,46
	$3D_2^{\triangle}$	2,6	8,7	3,30	4,9	0,89	1,85	2,9	0,75	1,10	2,3	0,64	0,86
	$3D_3^{\triangle}$	23,9	330,9	13,87	184,8	0,90	7,75	107,9	0,77	4,53	93,4	0,59	3,92
	$3D_4^{\triangle}$	566,2	12 069,5	21,32	6 506,3	0,93	11,49	3 771,0	0,80	6,66	3 306,2	0,61	5,84
	$3D_5^{\triangle}$	37 695,3	(nepočítáno na CPU)										

(b) čtyřstěny

Tabulka 5.5: Srovnání doby výpočtu CT, efektivity Eff paralelního výpočtu na CPU a urychlení GSp výpočtu na GPU při výpočtu úlohy popsané v sekci 5.4 ve 3D.

5.5 Kompoziční proudění

Druhou testovací úlohou je úloha kompozičního proudění za použití rovnovážného přístupu pro popis přestupu komponent mezi fázemi, který byl popsán v sekci 1.9. Matematický model vychází z rovnic dvoufázového proudění (1.16), ke kterým je přidána transportní rovnice (1.20) pro jednu komponentu

rozpuštěnou v obou fázích. Koeficienty vyjadřující tuto úlohu pomocí obecné soustavy parciálních diferenciálních rovnic (2.1) jsou uvedeny v sekci 2.2.

Úlohu řešíme opět ve 2D a 3D ve stejné geometrické konfiguraci jako na obr. 5.1. Pro první dvě rovnice pro neznámé $Z_1 = p_w$ a $Z_2 = p_n$ použijeme stejné nastavení jako v sekci 5.4. Díky použití rovnovážného přístupu třetí rovnice pro neznámou $Z_3 = X$ neovlivňuje řešení prvních dvou rovnic, tedy ze sekce 5.4 můžeme převzít i semi-analytické řešení pro první dvě rovnice. Při konstrukci analytického řešení třetí rovnice předpokládáme řešení X ve sférických souřadnicích ve tvaru

$$X(\rho, t) = X_0 \exp\left(-B_X \rho^2 \mathrm{e}^{-A_X t}\right), \tag{5.11}$$

kde $\rho = \|\boldsymbol{x}\|$ je radiální souřadnice, $X_0[-]$ je hmotnostní zlomek komponenty v bodě $\boldsymbol{x} = \mathbf{0}$ a $A_X[\mathrm{s}^{-1}]$ a $B_X[\mathrm{m}^{-2}]$ jsou kladné konstanty. Analytická funkce daná rovnicí (5.11) splňuje rovnici (1.20), pokud reakční člen r_X má ve sférických souřadnicích tvar

$$r_X(\rho, t) = -B_X \left(\phi \rho^2 A_X + 2 D_X d - 2\rho v_t(\rho, t) - 4 D_X \rho^2 B_X \mathrm{e}^{-A_X t}\right) \mathrm{e}^{-A_X t}, \tag{5.12}$$

kde $v_t(\rho, t)$ je radiální složka rychlosti $\boldsymbol{v}_t$.

Jako počáteční podmínka pro třetí rovnici slouží funkce (5.11) vyčíslená v čase $t = 0$. Okrajové podmínky pro neznámou X volíme podobně jako pro první dvě neznámé. V počátku souřadného systému a na částech hranice reprezentujících podmínky v nekonečnu (tj. Γ_N a Γ_E ve 2D a přední stěny ve 3D, viz obr. 5.1) předepisujeme Dirichletovu podmínku danou přesným řešením (5.11) a na zbývajících částech hranice předepisujeme nulovou Neumannovu podmínku.

Úlohu řešíme s rovnicí pro neznámou X v konzervativním tvaru (1.20) i v nekonzervativním tvaru (2.7). Dále uvažujeme rozklad difúzního koeficientu $D_X = m_X D_0$ popsaný v sekci 2.2 a volíme konstantní difúzní koeficient $D_0 = 10^{-5}\,\mathrm{m}^2\mathrm{s}^{-1}$ a koeficient mobility $m_X = 0$ nebo $m_X = 1$. Dále volíme koeficienty $A_X = 5 \cdot 10^{-5}\,\mathrm{s}^{-1}$ a $B_X = 20\,\mathrm{m}^{-2}$.

Výsledky numerické analýzy pomocí L^1 normy chyby $E_{h,X}$ jsou uvedeny v tab. 5.6 pro model Brookse a Coreyho a v tab. 5.7 pro model van Genuchtena. Volba časových kroků pro jednotlivé sítě je uvedena v tab. 5.1. Ve 2D stačí volit časové kroky stejně jako v úloze dvoufázového proudění, ve 3D je však potřeba při zjemňování sítě zkracovat časový krok mnohem rychleji oproti předchozí úloze. To je nejspíš způsobeno chybou časové diskretizace toku Q_0, který je dle vztahu (5.8) pro $d = 3$ úměrný $\sqrt{t}$, což se projeví v rovnici (1.20) ve členech závisejících na totální rychlosti $\boldsymbol{v}_t$. Ve všech případech experimentální řády konvergence poukazují na konvergenci numerického schématu s prvním řádem přesnosti v L^1 normě. Ačkoli v čistě advekčním případě jsou normy chyby $E_{h,X}$ podstatně větší oproti difúznímu případu, stabilizace numerického schématu pomocí upwindu se jeví jako dostatečná.

Analýzu efektivity výpočtu pro kompoziční úlohu provedeme pro model Brookse a Coreyho, konzervativní tvar rovnice (1.20) a koeficient mobility $m_X = 0$. Výpočty probíhaly na stejném hardware jako v případě dvoufázové úlohy popsané v předchozí sekci, pro kterou jsou výsledky analýzy efektivity výpočtu uvedeny v tab. 5.4 a 5.5. Z důvodu omezené výpočetní kapacity pracoviště KM FJFI však v případě kompoziční úlohy porovnáme pouze dobu výpočtu CT a urychlení GSp pro výpočty na GPU a 6 vláknech CPU. Výsledky uvedené v tab. 5.8 ukazují, že výpočet na GPU je pro jemné sítě přibližně $5\times$ rychlejší oproti šestivláknovému výpočtu na CPU, což odpovídá výsledkům pro dvoufázovou úlohu. V případě dvojrozměrných sítí, kde byly časové kroky voleny stejně jako v případě dvoufázové úlohy, je prodloužení doby výpočtu přibližně úměrné rozdílu mezi počty stupňů volnosti N_{dof} v obou úlohách. V případě trojrozměrných sítí bylo potřeba volit časové kroky jinak a proto nelze přímo porovnat příslušné doby výpočtu pro dvoufázovou a kompoziční úlohu.

Id.	Konzervativní tvar				Nekonzervativní tvar			
	$D_X = 0\,\mathrm{m}^2/\mathrm{s}$		$D_X = 10^{-5}\,\mathrm{m}^2/\mathrm{s}$		$D_X = 0\,\mathrm{m}^2/\mathrm{s}$		$D_X = 10^{-5}\,\mathrm{m}^2/\mathrm{s}$	
	$\|E_{h,X}\|_1$	$eoc_{X,1}$	$\|E_{h,X}\|_1$	$eoc_{X,1}$	$\|E_{h,X}\|_1$	$eoc_{X,1}$	$\|E_{h,X}\|_1$	$eoc_{X,1}$
$2D_1^\square$	$2{,}59 \cdot 10^{-2}$		$8{,}46 \cdot 10^{-3}$		$2{,}59 \cdot 10^{-2}$		$8{,}46 \cdot 10^{-3}$	
		1,08		**1,01**		**1,08**		**1,01**
$2D_2^\square$	$1{,}23 \cdot 10^{-2}$		$4{,}19 \cdot 10^{-3}$		$1{,}23 \cdot 10^{-2}$		$4{,}19 \cdot 10^{-3}$	
		0,99		**0,99**		**0,99**		**0,99**
$2D_3^\square$	$6{,}18 \cdot 10^{-3}$		$2{,}11 \cdot 10^{-3}$		$6{,}18 \cdot 10^{-3}$		$2{,}11 \cdot 10^{-3}$	
		0,97		**0,97**		**0,97**		**0,97**
$2D_4^\square$	$3{,}16 \cdot 10^{-3}$		$1{,}08 \cdot 10^{-3}$		$3{,}16 \cdot 10^{-3}$		$1{,}08 \cdot 10^{-3}$	
		0,95		**0,94**		**0,95**		**0,94**
$2D_5^\square$	$1{,}64 \cdot 10^{-3}$		$5{,}59 \cdot 10^{-4}$		$1{,}64 \cdot 10^{-3}$		$5{,}59 \cdot 10^{-4}$	
$2D_1^\triangle$	$2{,}55 \cdot 10^{-2}$		$8{,}79 \cdot 10^{-3}$		$2{,}55 \cdot 10^{-2}$		$8{,}79 \cdot 10^{-3}$	
		1,41		**1,08**		**1,41**		**1,08**
$2D_2^\triangle$	$1{,}02 \cdot 10^{-2}$		$4{,}36 \cdot 10^{-3}$		$1{,}02 \cdot 10^{-2}$		$4{,}36 \cdot 10^{-3}$	
		0,92		**0,91**		**0,92**		**0,91**
$2D_3^\triangle$	$5{,}07 \cdot 10^{-3}$		$2{,}19 \cdot 10^{-3}$		$5{,}07 \cdot 10^{-3}$		$2{,}19 \cdot 10^{-3}$	
		1,09		**1,07**		**1,09**		**1,07**
$2D_4^\triangle$	$2{,}54 \cdot 10^{-3}$		$1{,}11 \cdot 10^{-3}$		$2{,}54 \cdot 10^{-3}$		$1{,}11 \cdot 10^{-3}$	
		0,97		**0,91**		**0,97**		**0,91**
$2D_5^\triangle$	$1{,}26 \cdot 10^{-3}$		$5{,}74 \cdot 10^{-4}$		$1{,}26 \cdot 10^{-3}$		$5{,}74 \cdot 10^{-4}$	
$3D_1^\square$	$9{,}35 \cdot 10^{-3}$		$3{,}45 \cdot 10^{-3}$		$9{,}40 \cdot 10^{-3}$		$3{,}45 \cdot 10^{-3}$	
		1,17		**1,00**		**1,18**		**1,00**
$3D_2^\square$	$4{,}14 \cdot 10^{-3}$		$1{,}72 \cdot 10^{-3}$		$4{,}14 \cdot 10^{-3}$		$1{,}72 \cdot 10^{-3}$	
		1,00		**0,95**		**1,00**		**0,95**
$3D_3^\square$	$2{,}07 \cdot 10^{-3}$		$8{,}89 \cdot 10^{-4}$		$2{,}07 \cdot 10^{-3}$		$8{,}89 \cdot 10^{-4}$	
$3D_1^\triangle$	$1{,}05 \cdot 10^{-2}$		$6{,}42 \cdot 10^{-3}$		$1{,}05 \cdot 10^{-2}$		$6{,}42 \cdot 10^{-3}$	
		0,66		**0,78**		**0,66**		**0,78**
$3D_2^\triangle$	$7{,}43 \cdot 10^{-3}$		$4{,}28 \cdot 10^{-3}$		$7{,}43 \cdot 10^{-3}$		$4{,}28 \cdot 10^{-3}$	
		1,08		**1,07**		**1,08**		**1,07**
$3D_3^\triangle$	$3{,}49 \cdot 10^{-3}$		$2{,}02 \cdot 10^{-3}$		$3{,}49 \cdot 10^{-3}$		$2{,}02 \cdot 10^{-3}$	
		1,25		**1,23**		**1,25**		**1,23**
$3D_4^\triangle$	$1{,}67 \cdot 10^{-3}$		$9{,}76 \cdot 10^{-4}$		$1{,}67 \cdot 10^{-3}$		$9{,}76 \cdot 10^{-4}$	

Tabulka 5.6: Výsledky numerické analýzy pomocí L_1 normy chyby $E_{h,X}$ pro model Brookse a Coreyho.

Id.	Konzervativní tvar				Nekonzervativní tvar			
	$D_X = 0\,\mathrm{m}^2/\mathrm{s}$		$D_X = 10^{-5}\,\mathrm{m}^2/\mathrm{s}$		$D_X = 0\,\mathrm{m}^2/\mathrm{s}$		$D_X = 10^{-5}\,\mathrm{m}^2/\mathrm{s}$	
	$\|E_{h,X}\|_1$	$eoc_{X,1}$	$\|E_{h,X}\|_1$	$eoc_{X,1}$	$\|E_{h,X}\|_1$	$eoc_{X,1}$	$\|E_{h,X}\|_1$	$eoc_{X,1}$
$2D_1^\square$	$2{,}59 \cdot 10^{-2}$		$8{,}45 \cdot 10^{-3}$		$2{,}59 \cdot 10^{-2}$		$8{,}45 \cdot 10^{-3}$	
		1,09		**1,01**		**1,09**		**1,01**
$2D_2^\square$	$1{,}21 \cdot 10^{-2}$		$4{,}18 \cdot 10^{-3}$		$1{,}21 \cdot 10^{-2}$		$4{,}18 \cdot 10^{-3}$	
		1,00		**0,99**		**1,00**		**0,99**
$2D_3^\square$	$6{,}05 \cdot 10^{-3}$		$2{,}11 \cdot 10^{-3}$		$6{,}05 \cdot 10^{-3}$		$2{,}11 \cdot 10^{-3}$	
		0,99		**0,97**		**0,99**		**0,97**
$2D_4^\square$	$3{,}04 \cdot 10^{-3}$		$1{,}07 \cdot 10^{-3}$		$3{,}04 \cdot 10^{-3}$		$1{,}07 \cdot 10^{-3}$	
		0,99		**0,95**		**0,99**		**0,95**
$2D_5^\square$	$1{,}53 \cdot 10^{-3}$		$5{,}56 \cdot 10^{-4}$		$1{,}53 \cdot 10^{-3}$		$5{,}56 \cdot 10^{-4}$	
$2D_1^\triangle$	$2{,}08 \cdot 10^{-2}$		$8{,}88 \cdot 10^{-3}$		$2{,}08 \cdot 10^{-2}$		$8{,}88 \cdot 10^{-3}$	
		1,11		**1,09**		**1,11**		**1,09**
$2D_2^\triangle$	$1{,}01 \cdot 10^{-2}$		$4{,}36 \cdot 10^{-3}$		$1{,}01 \cdot 10^{-2}$		$4{,}36 \cdot 10^{-3}$	
		0,91		**0,91**		**0,91**		**0,91**
$2D_3^\triangle$	$5{,}05 \cdot 10^{-3}$		$2{,}19 \cdot 10^{-3}$		$5{,}05 \cdot 10^{-3}$		$2{,}19 \cdot 10^{-3}$	
		1,09		**1,07**		**1,09**		**1,07**
$2D_4^\triangle$	$2{,}54 \cdot 10^{-3}$		$1{,}11 \cdot 10^{-3}$		$2{,}54 \cdot 10^{-3}$		$1{,}11 \cdot 10^{-3}$	
		0,97		**0,91**		**0,97**		**0,91**
$2D_5^\triangle$	$1{,}26 \cdot 10^{-3}$		$5{,}74 \cdot 10^{-4}$		$1{,}26 \cdot 10^{-3}$		$5{,}74 \cdot 10^{-4}$	
$3D_1^\square$	$9{,}57 \cdot 10^{-3}$		$3{,}45 \cdot 10^{-3}$		$9{,}51 \cdot 10^{-3}$		$3{,}45 \cdot 10^{-3}$	
		1,21		**1,00**		**1,20**		**1,00**
$3D_2^\square$	$4{,}14 \cdot 10^{-3}$		$1{,}72 \cdot 10^{-3}$		$4{,}14 \cdot 10^{-3}$		$1{,}72 \cdot 10^{-3}$	
		1,00		**0,95**		**1,00**		**0,95**
$3D_3^\square$	$2{,}07 \cdot 10^{-3}$		$8{,}89 \cdot 10^{-4}$		$2{,}07 \cdot 10^{-3}$		$8{,}89 \cdot 10^{-4}$	
$3D_1^\triangle$	$1{,}05 \cdot 10^{-2}$		$6{,}42 \cdot 10^{-3}$		$1{,}05 \cdot 10^{-2}$		$6{,}42 \cdot 10^{-3}$	
		0,66		**0,78**		**0,66**		**0,78**
$3D_2^\triangle$	$7{,}43 \cdot 10^{-3}$		$4{,}28 \cdot 10^{-3}$		$7{,}43 \cdot 10^{-3}$		$4{,}28 \cdot 10^{-3}$	
		1,08		**1,07**		**1,08**		**1,07**
$3D_3^\triangle$	$3{,}49 \cdot 10^{-3}$		$2{,}02 \cdot 10^{-3}$		$3{,}49 \cdot 10^{-3}$		$2{,}02 \cdot 10^{-3}$	
		1,25		**1,23**		**1,25**		**1,23**
$3D_4^\triangle$	$1{,}67 \cdot 10^{-3}$		$9{,}76 \cdot 10^{-4}$		$1{,}67 \cdot 10^{-3}$		$9{,}76 \cdot 10^{-4}$	

Tabulka 5.7: Výsledky numerické analýzy pomocí L_1 normy chyby $E_{h,X}$ pro model van Genuchtena.

	Id.	GPU CT	CPU (6 vláken) CT	CPU (6 vláken) GSp
MGSR	$2D_1^{\square}$	5,0	0,8	0,15
	$2D_2^{\square}$	29,0	6,4	0,22
	$2D_3^{\square}$	114,2	57,9	0,51
	$2D_4^{\square}$	846,1	1 466,5	1,73
	$2D_5^{\square}$	10 798,1	26 632,8	2,47
CWY	$2D_1^{\square}$	1,7	0,3	0,21
	$2D_2^{\square}$	11,5	4,9	0,42
	$2D_3^{\square}$	48,7	61,2	1,26
	$2D_4^{\square}$	477,8	1 757,6	3,68
	$2D_5^{\square}$	6 507,0	35 661,5	5,48

(a) obdélníky

	Id.	GPU CT	CPU (6 vláken) CT	CPU (6 vláken) GSp
MGSR	$2D_1^{\triangle}$	4,5	0,7	0,15
	$2D_2^{\triangle}$	21,7	4,1	0,19
	$2D_3^{\triangle}$	119,0	49,6	0,42
	$2D_4^{\triangle}$	838,6	1 016,3	1,21
	$2D_5^{\triangle}$	9 032,4	19 089,4	2,11
CWY	$2D_1^{\triangle}$	1,4	0,3	0,20
	$2D_2^{\triangle}$	8,3	2,9	0,35
	$2D_3^{\triangle}$	50,9	49,5	0,97
	$2D_4^{\triangle}$	446,6	1 221,5	2,74
	$2D_5^{\triangle}$	5 066,7	26 584,9	5,25

(b) trojúhelníky

	Id.	GPU CT	CPU (6 vláken) CT	CPU (6 vláken) GSp
MGSR	$3D_1^{\square}$	10,3	7,6	0,73
	$3D_2^{\square}$	732,4	1 831,7	2,50
	$3D_3^{\square}$	72 208,4	(nepočítáno)	
CWY	$3D_1^{\square}$	4,4	8,7	1,96
	$3D_2^{\square}$	458,2	2 376,8	5,19
	$3D_3^{\square}$	52 700,1	(nepočítáno)	

(c) kvádry

	Id.	GPU CT	CPU (6 vláken) CT	CPU (6 vláken) GSp
MGSR	$3D_1^{\triangle}$	201,5	50,8	0,25
	$3D_2^{\triangle}$	572,6	289,0	0,50
	$3D_3^{\triangle}$	3 320,1	6 574,0	1,98
	$3D_4^{\triangle}$	72 146,0	337 187,0	4,67
CWY	$3D_1^{\triangle}$	70,2	42,8	0,61
	$3D_2^{\triangle}$	251,6	308,6	1,23
	$3D_3^{\triangle}$	1 936,3	8 179,5	4,22
	$3D_4^{\triangle}$	49 641,9	284 828,0	5,74

(d) čtyřstěny

Tabulka 5.8: Srovnání doby výpočtu CT a urychlení GSp výpočtu na GPU oproti CPU při výpočtu úlohy popsané v sekci 5.5. Pro výpočet byl použit model Brookse a Coreyho, konzervativní tvar rovnice (1.20) a koeficient mobility $m_X = 0$.

Závěr

V této práci jsme se zabývali matematickým modelováním dvoufázového kompozičního proudění v porézním prostředí. V první kapitole je popsán matematický model nemísivého dvoufázového proudění ovlivněného kapilaritou a model kompozičního proudění využívající rovnovážný přístup pro popis přestupu komponent mezi fázemi. Pro oba modely je zformulována soustava parciálních diferenciálních rovnic popisující příslušný model.

Numerické schéma je odvozeno v prostoru obecné dimenze pro soustavu n parciálních diferenciálních rovnic s obecnými koeficienty, jejichž volbou lze schéma aplikovat na mnoho úloh, včetně modelů nemísivého a kompozičního dvoufázového proudění v porézním prostředí. Semi-implicitní numerické schéma je založeno na kombinaci hybridní metody smíšených konečných prvků a nespojité Galerkinovy metody, v obou případech používáme prostory nejnižšího řádu pro aproximaci neznámých funkcí. Ke stabilizaci schématu byla použita technika upwind a pro sítě tvořené úsečkami, obdélníky nebo kvádry byla dále použita metoda mass-lumping, která je velmi důležitá pro stabilizaci řešení na materiálových rozhraních v heterogenním prostředí.

Numerické schéma bylo použito pro implementaci hybridního řešiče s pomocí knihovny `TNL`. Výpočet lze spustit buď paralelně na vícejádrovém CPU pomocí OpenMP nebo masivně paralelně na GPU pomocí platformy CUDA. Efektivnímu využití obou výpočetních architektur je třeba přizpůsobit návrh algoritmů a rozložení dat v paměti, testovací úloha popsaná ve čtvrté kapitole ukazuje několikanásobný rozdíl mezi optimálním a neoptimálním přístupem do paměti. Nejnáročnější částí výpočetního algoritmu je řešení velkých soustav s řídkou maticí, což řešíme pomocí iterační metody GMRES s adaptivní volbou restartovacího parametru. Pro zlepšení efektivity výpočtu na GPU jsme navrhli modifikaci metody GMRES využívající ortogonalizaci pomocí Householderových transformací a kompaktní WY reprezentace. Poslední sekce čtvrté kapitoly se věnuje implementaci nestrukturovaných sítí v knihovně `TNL` a souvisejícím optimalizacím pro různé výpočetní architektury. Na testovací úloze je ukázáno, že podstatný vliv na efektivitu výpočtu má očíslování elementů sítě, na CPU je rozdíl mezi efektivním a neefektivním očíslováním až 28% a na GPU až 13%.

V poslední kapitole jsou popsány testovací úlohy nemísivého a kompozičního dvoufázového proudění, které sloužily k verifikaci numerického schématu pomocí analýzy experimentálního rádu konvergence. Numerické simulace byly provedeny ve dvojrozměrném a trojrozměrném prostoru za použití strukturovaných sítí tvořených obdélníky nebo kvádry a nestrukturovaných sítí tvořených trojúhelníky nebo čtyřstěny. Kompoziční úloha byla analyzována pro konzervativní a nekonzervativní tvar transportní rovnice a s různými volbami difúzního koeficientu pro ověření chování schématu v advekčním a difúzním případě. Výsledky numerické analýzy ukazují, že ve všech případech numerické schéma konverguje s prvním řádem přesnosti. Výsledky analýzy efektivity paralelního výpočtu ukazují, že výpočet na CPU dobře škáluje až na nejvyšší dostupný počet jader procesoru a na GPU je výpočet pro dostatečně velké strukturované sítě až 20× (ve 2D) a až 25× (ve 3D) rychlejší oproti sekvenčnímu výpočtu na CPU a až 5× (ve 2D) a až 7× (ve 3D) rychlejší oproti výpočtu na 6 vláknech CPU. Pro nestrukturované sítě je urychlení výpočtu pomocí GPU mírně menší oproti strukturovaným sítím, ale při zjemňování sítí urychlení stále roste a také paralelní výpočet na CPU dobře škáluje.

Formulace numerického schématu a velká část výsledků uvedených v této práci byly odeslány k recenzi v článku [16]. Hlavním dodatečným a zatím nepublikovaným přínosem této práce je možnost použití nestrukturovaných sítí pro masivně paralelní výpočty na GPU.

Příloha A

Výpočet prvků lokálních matic

Numerické schéma popsané v kapitole 3 doplníme o podrobnosti týkající se výpočtu prvků lokálních matic $\mathbf{b}_{i,j,K} = [b_{i,j,K,E,F}]_{E,F\in\mathcal{E}_K}$ definovaných pro všechna $i,j = 1,\dots,n$ a $K \in \mathcal{K}_h$ v sekci 3.1.4. Při výpočtu se omezíme na případ, kdy tenzor $\mathbf{D}_{i,j} = D_{i,j}\mathbf{I}$ popisuje izotropní prostředí a střední hodnotu $D_{i,j}$ na elementu K označíme jako $D_{i,j,K}$.

A.1 Matice $\mathrm{b}_{i,j,K}$ pro úsečky

Uvažujme referenční element $K = (0, h_x) \subset \mathbb{R}^1$ a symboly E_1, resp. E_2 označíme levou, resp. pravou hranu elementu K. Přímým výpočtem definičních integrálů $B_{i,j,K,E,F} = D_{i,j,K}^{-1}\int_K \boldsymbol{\omega}_{K,E}^T\boldsymbol{\omega}_{K,F}$ pro $E,F \in \mathcal{E}_K$ dostaneme

$$\mathbf{B}_{i,j,K} = \frac{h_x}{6}D_{i,j,K}^{-1}\begin{pmatrix} 2 & -1 \\ -1 & 2 \end{pmatrix}. \tag{A.1}$$

Inverzní matice $\mathbf{b}_{i,j,K} = \mathbf{B}_{i,j,K}^{-1}$ má tvar

$$\mathbf{b}_{i,j,K} = \frac{2}{h_x}D_{i,j,K}\begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}. \tag{A.2}$$

V praxi se však ukazuje (viz [16, 28]), že tato volba koeficientů $b_{i,j,K,E,F}$ může při simulaci úloh v heterogenním prostředí způsobovat nežádoucí oscilace. Teoretické zdůvodnění tohoto jevu dle [51] spočívá v tom, že výsledná matice soustavy pro MHFEM není při použití obdélníkových nebo kvádrových elementů M-matice. Řešením je použít metodu *mass-lumping* a integrály definující prvky matice $\mathbf{B}_{i,j,K}$ spočítat pouze přibližně pomocí numerické integrace

$$\int_K \xi(\boldsymbol{x})\mathrm{d}\boldsymbol{x} \approx \frac{|K|}{\#\mathcal{V}_K}\sum_{\iota=1}^{\#\mathcal{V}_K}\xi(\boldsymbol{x}_\iota), \tag{A.3}$$

kde ξ je integrovaná skalární funkce, $\mathcal{V}_K$ je množina vrcholů elementu K a body $\boldsymbol{x}_\iota$ umístíme do vrcholů elementu K. Takto získaná matice $\mathbf{B}_{i,j,K}^{(\ell)}$ je diagonální a má tvar

$$\mathbf{B}_{i,j,K} \approx \mathbf{B}_{i,j,K}^{(\ell)} = \frac{h_x}{2}D_{i,j,K}^{-1}\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}. \tag{A.4}$$

Matici $\mathbf{b}_{i,j,K} = \mathbf{B}_{i,j,K}^{-1}$ pak aproximujeme inverzí matice $\mathbf{B}_{i,j,K}^{(\ell)}$, tedy

$$\mathbf{b}_{i,j,K} \approx \frac{2}{h_x}D_{i,j,K}\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}. \tag{A.5}$$

A.2 Matice $\mathrm{b}_{i,j,K}$ pro obdélníky

Uvažujme referenční element $K = (0, h_x) \times (0, h_y) \subset \mathbb{R}^2$ a stejně jako v sekci 3.1.1 označíme symboly E_1, E_2, E_3, E_4 postupně levou, pravou, dolní a horní hranu elementu K. Podobně jako v předchozí sekci dostaneme přímým výpočtem definičních integrálů matici $\mathbf{B}_{i,j,K}$ ve tvaru

$$\mathbf{B}_{i,j,K} = \frac{1}{6} D_{i,j,K}^{-1} \begin{pmatrix} 2\frac{h_x}{h_y} & -\frac{h_x}{h_y} & 0 & 0 \\ -\frac{h_x}{h_y} & 2\frac{h_x}{h_y} & 0 & 0 \\ 0 & 0 & 2\frac{h_y}{h_x} & -\frac{h_y}{h_x} \\ 0 & 0 & -\frac{h_y}{h_x} & 2\frac{h_y}{h_x} \end{pmatrix}. \tag{A.6}$$

Inverzní matice $\mathbf{b}_{i,j,K} = \mathbf{B}_{i,j,K}^{-1}$ má tvar

$$\mathbf{b}_{i,j,K} = 2 D_{i,j,K} \begin{pmatrix} 2\frac{h_y}{h_x} & \frac{h_y}{h_x} & 0 & 0 \\ \frac{h_y}{h_x} & 2\frac{h_y}{h_x} & 0 & 0 \\ 0 & 0 & 2\frac{h_x}{h_y} & \frac{h_x}{h_y} \\ 0 & 0 & \frac{h_x}{h_y} & 2\frac{h_x}{h_y} \end{pmatrix}. \tag{A.7}$$

Stejně jako v $\mathbb{R}^1$ může tato volba koeficientů $b_{i,j,K,E,F}$ pro obdélníkové elementy v $\mathbb{R}^2$ způsobovat nežádoucí oscilace. Opět tedy použijeme metodu mass-lumping a při výpočtu prvků matice $B_{i,j,K,E,F}$ použijeme numerickou integraci (A.3), čímž získáme matici $\mathbf{B}_{i,j,K}^{(\ell)}$ ve tvaru

$$\mathbf{B}_{i,j,K} \approx \mathbf{B}_{i,j,K}^{(\ell)} = \frac{1}{2} D_{i,j,K}^{-1} \begin{pmatrix} \frac{h_x}{h_y} & 0 & 0 & 0 \\ 0 & \frac{h_x}{h_y} & 0 & 0 \\ 0 & 0 & \frac{h_y}{h_x} & 0 \\ 0 & 0 & 0 & \frac{h_y}{h_x} \end{pmatrix}. \tag{A.8}$$

Matici $\mathbf{b}_{i,j,K} = \mathbf{B}_{i,j,K}^{-1}$ pak aproximujeme inverzí matice $\mathbf{B}_{i,j,K}^{(\ell)}$, tedy

$$\mathbf{b}_{i,j,K} \approx 2 D_{i,j,K} \begin{pmatrix} \frac{h_y}{h_x} & 0 & 0 & 0 \\ 0 & \frac{h_y}{h_x} & 0 & 0 \\ 0 & 0 & \frac{h_x}{h_y} & 0 \\ 0 & 0 & 0 & \frac{h_x}{h_y} \end{pmatrix}. \tag{A.9}$$

A.3 Matice $\mathbf{b}_{i,j,K}$ pro kvádry

Uvažujme referenční element $K = (0, h_x) \times (0, h_y) \times (0, h_z) \subset \mathbb{R}^3$ a stěny elementu $E_\iota \in \mathcal{E}_K$ očíslujeme stejně jako v sekci 3.1.1. Podobně jako v předchozích sekcích dostaneme přímým výpočtem definičních integrálů matici $\mathbf{B}_{i,j,K}$ ve tvaru

$$\mathbf{B}_{i,j,K} = \frac{1}{6} D_{i,j,K}^{-1} \begin{pmatrix} 2\frac{h_x}{h_y h_z} & -\frac{h_x}{h_y h_z} & 0 & 0 & 0 & 0 \\ -\frac{h_x}{h_y h_z} & 2\frac{h_x}{h_y h_z} & 0 & 0 & 0 & 0 \\ 0 & 0 & 2\frac{h_y}{h_x h_z} & -\frac{h_y}{h_x h_z} & 0 & 0 \\ 0 & 0 & -\frac{h_y}{h_x h_z} & 2\frac{h_y}{h_x h_z} & 0 & 0 \\ 0 & 0 & 0 & 0 & 2\frac{h_z}{h_x h_y} & -\frac{h_z}{h_x h_y} \\ 0 & 0 & 0 & 0 & -\frac{h_z}{h_x h_y} & 2\frac{h_z}{h_x h_y} \end{pmatrix}. \tag{A.10}$$

Inverzní matice $\mathbf{b}_{i,j,K} = \mathbf{B}_{i,j,K}^{-1}$ má tvar

$$\mathbf{b}_{i,j,K} = 2 D_{i,j,K} \begin{pmatrix} 2\frac{h_y h_z}{h_x} & \frac{h_y h_z}{h_x} & 0 & 0 & 0 & 0 \\ \frac{h_y h_z}{h_x} & 2\frac{h_y h_z}{h_x} & 0 & 0 & 0 & 0 \\ 0 & 0 & 2\frac{h_x h_z}{h_y} & \frac{h_x h_z}{h_y} & 0 & 0 \\ 0 & 0 & \frac{h_x h_z}{h_y} & 2\frac{h_x h_z}{h_y} & 0 & 0 \\ 0 & 0 & 0 & 0 & 2\frac{h_x h_y}{h_z} & \frac{h_x h_y}{h_z} \\ 0 & 0 & 0 & 0 & \frac{h_x h_y}{h_z} & 2\frac{h_x h_y}{h_z} \end{pmatrix}. \tag{A.11}$$

Stejně jako v $\mathbb{R}^1$ může tato volba koeficientů $b_{i,j,K,E,F}$ pro kvádrové elementy v $\mathbb{R}^3$ způsobovat nežádoucí oscilace. Opět tedy použijeme metodu mass-lumping a při výpočtu prvků matice $B_{i,j,K,E,F}$ použijeme numerickou integraci (A.3), čímž získáme matici $\mathbf{B}_{i,j,K}^{(\ell)}$ ve tvaru

$$\mathbf{B}_{i,j,K} \approx \mathbf{B}_{i,j,K}^{(\ell)} = \frac{1}{2} D_{i,j,K}^{-1} \begin{pmatrix} \frac{h_x}{h_y h_z} & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{h_x}{h_y h_z} & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{h_y}{h_x h_z} & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{h_y}{h_x h_z} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{h_z}{h_y h_z} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{h_z}{h_y h_z} \end{pmatrix}. \tag{A.12}$$

Matici $\mathbf{b}_{i,j,K} = \mathbf{B}_{i,j,K}^{-1}$ pak aproximujeme inverzí matice $\mathbf{B}_{i,j,K}^{(\ell)}$, tedy

$$\mathbf{b}_{i,j,K} \approx 2 D_{i,j,K} \begin{pmatrix} \frac{h_y h_z}{h_x} & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{h_y h_z}{h_x} & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{h_x h_z}{h_y} & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{h_x h_z}{h_y} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{h_x h_y}{h_z} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{h_x h_y}{h_z} \end{pmatrix}. \tag{A.13}$$

A.4 Matice $\mathbf{b}_{i,j,K}$ pro trojúhelníky

Uvažujme trojúhelník $K = [\boldsymbol{V}_0, \boldsymbol{V}_1, \boldsymbol{V}_2]_\kappa \subset \mathbb{R}^2$ a jeho hrany $E_\iota \in \mathcal{E}_K$ očíslujeme stejně jako v sekci 3.1.1. Pro výpočet integrálů $B_{i,j,K,E_e,E_f} = D_{i,j,K}^{-1} \int_K \boldsymbol{\omega}_{K,E_e}^T \boldsymbol{\omega}_{K,E_f}$ pro $e, f = 0, 1, 2$ zavedeme barycentrické souřadnice $\eta_0, \eta_1 \in [0, 1]$, $\eta_0 + \eta_1 \leq 1$. Při označení $\boldsymbol{P}_e = \boldsymbol{V}_e - \boldsymbol{V}_2$, $e = 0, 1, 2$ můžeme každý bod $\boldsymbol{x} \in K$ zapsat ve tvaru

$$\boldsymbol{x} = \boldsymbol{V}_2 + \eta_0 \boldsymbol{P}_0 + \eta_1 \boldsymbol{P}_1. \tag{A.14}$$

Absolutní hodnota Jakobiánu této transformace je

$$\begin{aligned} |\mathcal{J}| &= \left| \det \begin{pmatrix} P_0^{(1)} & P_1^{(1)} \\ P_0^{(2)} & P_1^{(2)} \end{pmatrix} \right| = \left| P_0^{(1)} P_1^{(2)} - P_0^{(2)} P_1^{(1)} \right| \\ &= \left| (V_0^{(1)} - V_2^{(1)})(V_1^{(2)} - V_2^{(2)}) - (V_0^{(2)} - V_2^{(2)})(V_1^{(1)} - V_2^{(1)}) \right| = 2|K|_2. \end{aligned} \tag{A.15}$$

Použitím této transformace pro výpočet integrálu dostaneme

$$\begin{aligned} \int_K \boldsymbol{\omega}_{K,E_e}^T \boldsymbol{\omega}_{K,E_f} \mathrm{d}\boldsymbol{x} &= \frac{1}{4|K|_2^2} \int_K (\boldsymbol{x} - \boldsymbol{V}_e)^T (\boldsymbol{x} - \boldsymbol{V}_f) \mathrm{d}\boldsymbol{x} \\ &= \frac{1}{2|K|_2} \int_0^1 \mathrm{d}\eta_1 \int_0^{1-\eta_1} \mathrm{d}\eta_0 (\eta_0 \boldsymbol{P}_0 + \eta_1 \boldsymbol{P}_1 - \boldsymbol{P}_e)^T (\eta_0 \boldsymbol{P}_0 + \eta_1 \boldsymbol{P}_1 - \boldsymbol{P}_f) \\ &= \frac{1}{2|K|_2} \int_0^1 \mathrm{d}\eta_1 \int_0^{1-\eta_1} \mathrm{d}\eta_0 \Big(\eta_0^2 \boldsymbol{P}_0^T \boldsymbol{P}_0 + \eta_1^2 \boldsymbol{P}_1^T \boldsymbol{P}_1 + \boldsymbol{P}_e^T \boldsymbol{P}_f \\ &\qquad + 2\eta_0\eta_1 \boldsymbol{P}_0^T \boldsymbol{P}_1 - \eta_0 \boldsymbol{P}_0^T (\boldsymbol{P}_e + \boldsymbol{P}_f) - \eta_1 \boldsymbol{P}_1^T (\boldsymbol{P}_e + \boldsymbol{P}_f) \Big) \\ &= \frac{1}{24|K|_2} \left(\boldsymbol{P}_0^T \boldsymbol{P}_0 + \boldsymbol{P}_1^T \boldsymbol{P}_1 + \boldsymbol{P}_0^T \boldsymbol{P}_1 - 2(\boldsymbol{P}_0 + \boldsymbol{P}_1)^T (\boldsymbol{P}_e + \boldsymbol{P}_f) + 6 \boldsymbol{P}_e \boldsymbol{P}_f \right). \end{aligned} \tag{A.16}$$

Všechny prvky matice $\mathbf{B}_{i,j,K}$ jsou obecně nenulové a mass-lumping pomocí numerické integrace (A.3) nelze použít. Inverzní matici $\mathbf{b}_{i,j,K}$ počítáme algoritmicky postupem popsaným v sekci 4.2.1.

A.5 Matice $\mathbf{b}_{i,j,K}$ pro čtyřstěny

Uvažujme čtyřstěn $K = [\boldsymbol{V}_0, \boldsymbol{V}_1, \boldsymbol{V}_2, \boldsymbol{V}_3]_\kappa \subset \mathbb{R}^3$ a jeho hrany $E_\iota \in \mathcal{E}_K$ očíslujeme stejně jako v sekci 3.1.1. Pro výpočet integrálů $B_{i,j,K,E_e,E_f} = D_{i,j,K}^{-1} \int_K \boldsymbol{\omega}_{K,E_e}^T \boldsymbol{\omega}_{K,E_f}$ pro $e, f = 0, \dots, 3$ zavedeme barycentrické souřadnice $\eta_0, \eta_1, \eta_2 \in [0,1]$, $\eta_0 + \eta_1 + \eta_2 \leq 1$. Při označení $\boldsymbol{P}_e = \boldsymbol{V}_e - \boldsymbol{V}_3$, $e = 0, \dots, 3$ můžeme každý bod $\boldsymbol{x} \in K$ zapsat ve tvaru

$$\boldsymbol{x} = \boldsymbol{V}_3 + \eta_0 \boldsymbol{P}_0 + \eta_1 \boldsymbol{P}_1 + \eta_2 \boldsymbol{P}_2. \tag{A.17}$$

Absolutní hodnota Jakobiánu této transformace je

$$|\mathcal{J}| = \left| \det \begin{pmatrix} P_0^{(1)} & P_1^{(1)} & P_2^{(1)} \\ P_0^{(2)} & P_1^{(2)} & P_2^{(2)} \\ P_0^{(3)} & P_1^{(3)} & P_2^{(3)} \end{pmatrix} \right| = \left| \boldsymbol{P}_0 \cdot (\boldsymbol{P}_1 \times \boldsymbol{P}_2) \right| = 6|K|_3. \tag{A.18}$$

Použitím této transformace pro výpočet integrálu dostaneme

$$\begin{aligned}
\int_K \boldsymbol{\omega}_{K,E_e}^T \boldsymbol{\omega}_{K,E_f} \mathrm{d}\boldsymbol{x} &= \frac{1}{9|K|_3^2} \int_K (\boldsymbol{x} - \boldsymbol{V}_e)^T (\boldsymbol{x} - \boldsymbol{V}_f) \mathrm{d}\boldsymbol{x} \\
&= \frac{2}{3|K|_3} \int_0^1 \mathrm{d}\eta_2 \int_0^{1-\eta_2} \mathrm{d}\eta_1 \int_0^{1-\eta_1-\eta_2} \mathrm{d}\eta_0 \left(\sum_{k=0}^{2} \eta_k \boldsymbol{P}_k - \boldsymbol{P}_e \right)^T \left(\sum_{\ell=0}^{2} \eta_\ell \boldsymbol{P}_\ell - \boldsymbol{P}_f \right) \\
&= \frac{2}{3|K|_3} \int_0^1 \mathrm{d}\eta_2 \int_0^{1-\eta_2} \mathrm{d}\eta_1 \int_0^{1-\eta_1-\eta_2} \mathrm{d}\eta_0 \left(\sum_{k=0}^{2} \left(\eta_k^2 \boldsymbol{P}_k^T \boldsymbol{P}_k - \eta_k \boldsymbol{P}_k^T (\boldsymbol{P}_e + \boldsymbol{P}_f) \right) \right. \\
&\qquad \left. + \sum_{\substack{k,\ell=0 \\ k \neq \ell}}^{2} \eta_k \eta_\ell \boldsymbol{P}_k^T \boldsymbol{P}_\ell + \boldsymbol{P}_e^T \boldsymbol{P}_f \right) \\
&= \frac{1}{180|K|_3} \Big(2\boldsymbol{P}_0^T \boldsymbol{P}_0 + 2\boldsymbol{P}_1^T \boldsymbol{P}_1 + 2\boldsymbol{P}_2^T \boldsymbol{P}_2 + 2\boldsymbol{P}_0^T \boldsymbol{P}_1 + 2\boldsymbol{P}_0^T \boldsymbol{P}_2 + 2\boldsymbol{P}_1^T \boldsymbol{P}_2 \\
&\qquad - 5(\boldsymbol{P}_0 + \boldsymbol{P}_1 + \boldsymbol{P}_2)^T (\boldsymbol{P}_e + \boldsymbol{P}_f) + 20 \boldsymbol{P}_e \boldsymbol{P}_f \Big).
\end{aligned} \tag{A.19}$$

Všechny prvky matice $\mathbf{B}_{i,j,K}$ jsou obecně nenulové a mass-lumping pomocí numerické integrace (A.3) nelze použít. Inverzní matici $\mathbf{b}_{i,j,K}$ počítáme algoritmicky postupem popsaným v sekci 4.2.1.

Příloha B

Parametry materiálů a tekutin

Parametr			Písek A
Porozita	Φ	$[-]$	0,343
Vnitřní propustnost	K	$[\mathrm{m}^2]$	$5{,}168 \cdot 10^{-12}$
Reziduální saturace	S_{wr}	$[-]$	0,04
Reziduální saturace	S_{nr}	$[-]$	0
Brooksův a Coreyho parametr	$\lambda_{B\&C}$	$[-]$	5,408
Brooksův a Coreyho parametr	$p_{B\&C}$	$[\mathrm{Pa}]$	8 027,52
van Genuchtenův parametr	n_{vG}	$[-]$	12,49
van Genuchtenův parametr	α_{vG}	$[\mathrm{Pa}^{-1}]$	$1{,}08 \cdot 10^{-4}$

Tabulka B.1: Parametry porézního prostředí pro numerické simulace.

Parametr			Voda	NAPL
Hustota	ρ	$[\mathrm{kg} \cdot \mathrm{m}^{-1}]$	1 000	1 400
Dynamická viskozita	μ	$[\mathrm{Pa} \cdot \mathrm{s}]$	0,001	0,001

Tabulka B.2: Parametry tekutin pro numerické simulace.

Literatura

[1] A. H. Baker, E. R. Jessup, and T. V. Kolev, *A simple strategy for varying the restart parameter in GMRES(m)*, Journal of computational and applied mathematics 230.2 (2009), pages 751–761.

[2] P. Bastian, *Numerical computation of multiphase flow in porous media*, Habilitation Thesis, Univeristät Kiel, 1999.

[3] P. Bauer, V. Klement, T. Oberhuber, and V. Žabka, *Implementation of the Vanka-type multigrid solver for the finite element approximation of the Navier–Stokes equations on GPU*, Computer Physics Communications 200 (2016), pages 50–56.

[4] J. L. Bentley, *Multidimensional binary search trees used for associative searching*, Communications of the ACM 18.9 (1975), pages 509–517.

[5] F. Brezzi and M. Fortin, *Mixed and hybrid finite elements methods*, Springer series in computational mathematics, Springer-Verlag, 1991, ISBN: 978-1-4612-7824-5.

[6] R. H. Brooks and A. T. Corey, *Hydraulic properties of porous media and their relation to drainage design*, Transactions of the ASAE 7.1 (1964), pages 26–0028.

[7] N. T. Burdine, *Relative permeability calculations from pore size distribution data*, Journal of Petroleum Technology 5.03 (1953), pages 71–78.

[8] N. T. Burdine, L. S. Gournay, and P. P. Reichertz, *Pore size distribution of petroleum reservoir rocks*, Journal of Petroleum Technology 2.07 (1950), pages 195–204.

[9] Clawpack Development Team, *Clawpack software*, Version 5.4.0, 2017, DOI: 10.5281/zenodo.262111, URL: http://www.clawpack.org.

[10] L. Dagum and R. Menon, *OpenMP: an industry standard API for shared-memory programming*, Computational Science & Engineering, IEEE 5.1 (1998), pages 46–55.

[11] H. Darcy, *Les fontaines publiques de la ville de Dijon*, Victor Dalmont, Paris, 1856.

[12] T. A. Davis, *Algorithm 832: UMFPACK V4.3—an Unsymmetric-pattern Multifrontal Method*, ACM Trans. Math. Softw. 30.2 (June 2004), pages 196–199, ISSN: 0098-3500, DOI: 10.1145/992200.992206, URL: http://doi.acm.org/10.1145/992200.992206.

[13] C. J. van Duijn and M. J. De Neef, *Self-similar profiles for capillary diffusion driven flow in heterogeneous porous media*, Centrum voor Wiskunde en Informatica (1996).

[14] R. Fučík, *Advanced Numerical Methods for Modelling Two-Phase Flow in Heterogeneous Porous Media*, Dissertation Thesis, FNSPE, CTU in Prague, 2010.

[15] R. Fučík, T. H. Illangasekare, and M. Beneš, *Multidimensional self-similar analytical solutions of two-phase flow in porous media*, Advances in Water Resources 90 (2016), pages 51–56.

[16] R. Fučík, J. Klinkovský, J. Solovský, T. Oberhuber, and J. Mikyška, *Multidimensional mixed-hybrid finite element method for compositional two-phase flow in heterogeneous porous media and its parallel implementation on GPU*, Computer Physics Communications (under review).

[17] R. Fučík and J. Mikyška, *Discontinous Galerkin and Mixed-Hybrid Finite Element Approach to Two-Phase Flow in Heterogeneous Porous Media with Different Capillary Pressures*, Procedia Computer Science 4 (2011), pages 908–917.

[18] R. Fučík and J. Mikyška, *Mixed-hybrid finite element method for modelling two-phase flow in porous media*, Journal of Math-for-Industry 3.2 (2011), pages 9–19.

[19] R. Fučík, J. Mikyška, M. Beneš, and T. H. Illangasekare, *An improved semi-analytical solution for verification of numerical models of two-phase flow in porous media*, Vadose Zone Journal 6.1 (2007), pages 93–104.

[20] M. T. van Genuchten, *A closed-form equation for predicting the hydraulic conductivity of unsaturated soils*, Soil science society of America journal 44.5 (1980), pages 892–898.

[21] C. Geuzaine and J.-F. Remacle, *Gmsh: A 3-D finite element mesh generator with built-in pre- and post-processing facilities*, International Journal for Numerical Methods in Engineering 79.11 (2009), pages 1309–1331.

[22] L. Giraud, J. Langou, and M. Rozloznik, *The loss of orthogonality in the Gram-Schmidt orthogonalization process*, Computers & Mathematics with Applications 50.7 (2005), pages 1069–1075.

[23] R. Helmig, *Multiphase flow and transport processes in the subsurface: a contribution to the modeling of hydrosystems.* Springer-Verlag, 1997.

[24] P. Hénon, P. Ramet, and J. Roman, *PaStiX: A High-Performance Parallel Direct Solver for Sparse Symmetric Definite Systems*, Parallel Computing 28.2 (Jan. 2002), pages 301–321.

[25] H. Hoteit and A. Firoozabadi, *Numerical modeling of two-phase flow in heterogeneous permeable media with different capillarity pressures*, Advances in Water Resources 31.1 (2008), pages 56–73.

[26] Z. Chen, G. Huan, and Y. Ma, *Computational Methods for Multiphase Flows in Porous Media*, volume 2, SIAM, 2006.

[27] H. Ishigami, K. Kimura, and Y. Nakamura, *Implementation and performance evaluation of new inverse iteration algorithm with Householder transformation in terms of the compact WY representation*, Proc, 2011, pages 775–781.

[28] J. Klinkovský, *Numerické řešení dvoufázového nemísivého filtračního proudění s kapilaritou a jeho implementace na GPU*, Výzkumný úkol, FJFI, ČVUT v Praze, 2016.

[29] J. Klinkovský, *Numerické řešení jednofázového filtračního proudění a jeho implementace na GPU*, Bakalářská práce, FJFI, ČVUT v Praze, 2015.

[30] R. J. LeVeque, *Finite volume methods for hyperbolic problems*, volume 31, Cambridge university press, 2002.

[31] X. S. Li, *An Overview of SuperLU: Algorithms, Implementation, and User Interface*, 31.3 (Sept. 2005), pages 302–325.

[32] R. McKibbin, *Mathematical models for heat and mass transport in geothermal systems*, Transport phenomena in porous media, edited by D. B. Ingham and I. Pop, Elsevier, 1998.

[33] D. B. McWhorter and D. K. Sunada, *Exact integral solutions for two-phase flow*, Water Resources Research 26 (1990), pages 399–413.

[34] A. Monakov, A. Lokhmotov, and A. Avetisyan, *Automatically tuning sparse matrix-vector multiplication for GPU architectures*, High Performance Embedded Architectures and Compilers, Springer, 2010, pages 111–125.

[35] Y. Mualem, *A new model for predicting the hydraulic conductivity of unsaturated porous media*, Water resources research 12.3 (1976), pages 513–522.

[36] A. Muñoz, *Higher Order Ray Marching*, Computer Graphics Forum 33.8 (2014), pages 167–176, ISSN: 1467-8659, DOI: 10.1111/cgf.12424, URL: http://dx.doi.org/10.1111/cgf.12424.

[37] NVIDIA, *CUDA Toolkit Documentation, version 8.0*, 2017, URL: http://docs.nvidia.com/cuda/index.html.

[38] T. Oberhuber, J. Vacata, and A. Suzuki, *New Row-grouped CSR format for storing the sparse matrices on GPU with implementation in CUDA*, Acta Technica CSAV 56.4 (2011), pages 447–466.

[39] O. Pelech, *Pokročilé numerické metody pro dvoufázové proudění v nenasyceném porézním prostředí v 1D, 2D a 3D*, Diplomová práce, FJFI, ČVUT v Praze, 2016.

[40] C. G. Petra, O. Schenk, and M. Anitescu, *Real-time stochastic optimization of complex energy systems on high-performance computers*, IEEE Computing in Science & Engineering 16.5 (2014), pages 32–42.

[41] C. G. Petra, O. Schenk, M. Lubin, and K. Gärtner, *An augmented incomplete factorization approach for computing the Schur complement in stochastic optimization*, SIAM Journal on Scientific Computing 36.2 (2014), pages C139–C162.

[42] Y. Saad, *Iterative methods for sparse linear systems*, SIAM, 2003, ISBN: 0-89871-534-2.

[43] Y. Saad and M. H. Schultz, *GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems*, SIAM Journal on scientific and statistical computing 7.3 (1986), pages 856–869.

[44] R. Schreiber and C. Van Loan, *A storage-efficient WY representation for products of Householder transformations*, SIAM Journal on Scientific and Statistical Computing 10.1 (1989), pages 53–57.

[45] W. Schroeder, K. Martin, and B. Lorensen, *The Visualization Toolkit: An Object-oriented Approach to 3D Graphics*, Kitware, 2006, ISBN: 9781930934191.

[46] D. Vandevoorde and N. M. Josuttis, *C++ Templates: The Complete Guide*, Addison-Wesley Professional (2002).

[47] H. F. Walker, *Implementation of the GMRES method using Householder transformations*, SIAM Journal on Scientific and Statistical Computing 9.1 (1988), pages 152–163.

[48] J. Wu, A. Belevich, E. Bendersky, M. Heffernan, C. Leary, J. Pienaar, B. Roune, R. Springer, X. Weng, and R. Hundt, *GPUCC - An Open-Source GPGPU Compiler*, Proceedings of the 2016 International Symposium on Code Generation and Optimization, New York, NY, 2016, pages 105–116, URL: http://dl.acm.org/citation.cfm?id=2854041.

[49] W. A. Wulf and S. A. McKee, *Hitting the memory wall: implications of the obvious*, ACM SIGARCH computer architecture news 23.1 (1995), pages 20–24.

[50] Y. Yamamoto and Y. Hirota, *A parallel algorithm for incremental orthogonalization based on the compact WY representation*, JSIAM Letters 3.0 (2011), pages 89–92.

[51] A. Younes, P. Ackerer, and F. Lehmann, *A new mass lumping scheme for the mixed hybrid finite element method*, International journal for numerical methods in engineering 67.1 (2006), pages 89–107.