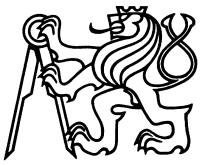
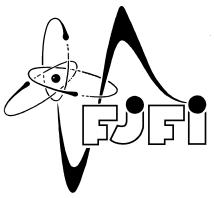


České vysoké učení technické v Praze
Fakulta jaderná a fyzikálně inženýrská



Numerické řešení jednofázového filtračního proudění a jeho implementace na GPU

Numerical solution of single-phase flow in porous media and its implementation on GPU

Bakalářská práce

Autor: **Jakub Klinkovský**

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

Akademický rok: 2014/2015

- Zadání práce -

Poděkování:
Děkuji svému školiteli Ing. Radku Fučíkovi, Ph.D. za pečlivost, ochotu a vstřícnost při vedení mé bakalářské práce a za řadu cenných připomínek, které napomohly ke vzniku této práce. Dále děkuji svému konzultantovi Ing. Tomáši Oberhuberovi, Ph.D. za neocenitelné rady při implementaci paralelního řešiče.

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

V Praze dne 3. července 2015 Jakub Klinkovský

Název práce: **Numerické řešení jednofázového filtračního proudění a jeho implementace na GPU**

Autor: Jakub Klinkovský

Obor: Matematické inženýrství

Zaměření: Matematické modelování

Druh práce: Bakalářská práce

Vedoucí práce: Ing. Radek Fučík, Ph.D., KM FJFI ČVUT

Abstrakt: Práce se zabývá masivně paralelní implementací na GPU hybridní metody smíšených prvků v problematice jednofázového stlačitelného proudění v porézním prostředí. Pro numerické řešení zformulovaného problému sestavíme numerické schéma založené na smíšené hybridní metodě konečných prvků a na semi-implicitním přístupu k časové diskretizaci. Výsledná soustava lineárních rovnic je řešena pomocí přímého nebo iteračního řešiče. Dále představíme paralelní implementaci numerického schématu s využitím knihovny `TNL` a architektury `CUDA`. Pomocí numerické analýzy ověříme konvergenci schématu a porovnáme přesnost a efektivitu implementovaných řešičů na CPU a GPU.

Klíčová slova: Jednofázové stlačitelné proudění, paralelizace, porézní prostředí, smíšená hybridní metoda konečných prvků

Title: **Numerical solution of single-phase flow in porous media and its implementation on GPU**

Author: Jakub Klinkovský

Abstract: The work deals with a massively parallel implementation on GPU of the mixed-hybrid finite element method for the mathematical modelling of a compressible single-phase flow in porous media. We devise a numerical scheme that is based on the mixed-hybrid finite element method with the semi-implicit approach for the time discretization in order to obtain a system of linear equations for which either a direct or iterative solver is used. We propose a parallel implementation of the numerical scheme using the `TNL` library and the `CUDA` architecture. We present a numerical analysis of the scheme and compare the efficiency and accuracy of its implementation on CPU and GPU.

Key words: Mixed-hybrid finite element method, parallelization, porous media, single-phase compressible flow

Obsah

Seznam zkratek

CFVM Cell-Centered Finite Volume Method 22
CSC Compressed Sparse Column 31
CSR Compressed Sparse Row 31
DOFs Degrees of Freedom 35
EOC Experimental Order of Convergence 46
FVM Finite Volume Method 14
GMRES Generalized Minimal Residual method 35
MFEM Mixed Finite Element Method 14
MHFEM Mixed Hybrid Finite Element Method 14
PU Processing Unit 51
REV Representative Elementary Volume 11

Úvod

Problematika studia procesů v porézním prostředí je velmi bohatá a 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, až po armádní aplikace jako např. detekce min. Pro studium proudění v porézním prostředí existuje mnoho matematických modelů 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 prvků a semi-implicitním přístupu k časové diskretizaci. Použití smíšené hybridní metody konečných prvků má oproti klasickým metodám řadu výhod: umožňuje použití obecných nestrukturovaných sítí, vede na soustavu s pozitivně definitní maticí a lze ji dobře paralelizovat. Linearizace v čase pomocí metody zamrzlých koeficientů vedoucí na semi-implicitní schéma má oproti plně implicitním metodám výhody v lepší efektivitě.

Cílem této práce je implementace masivně paralelního řešiče pro numerické schéma s využitím knihovny `TNL` a architektury `CUDA`. Přestože model jednofázového proudění představuje poměrně snadno numericky řešitelnou úlohu srovnatelnou s úlohou vedení tepla, náročná bude zejména paralelizace numerického schématu a implementace na GPU. Paralelizace na GPU je netriviální zejména z hlediska práce s pamětí a optimalizace kódu pro využití potenciálu specifické architektury dostupných GPU.

Výsledky této práce tedy jsou jednak sestavený matematický model a numerické schéma, zároveň však testovací kód využívající knihovnu `UMFPACK` na CPU, a především hlavní kód v jazyce C++ využívající knihovnu `TNL`, který je možné spustit jak na CPU, tak i masivně paralelně na GPU. Tento kód bude v budoucnu základem pro implementaci paralelního řešiče pro obecný model vícefázového kompozičního proudění v porézním prostředí.

Struktura práce

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

Ve druhé kapitole popíšeme numerické schéma použité pro řešení této soustavy doplněné o počáteční a okrajové podmínky. Prostorovou diskretizaci provedeme pomocí smíšené hybridní metody konečných prvků a metody konečných objemů. Pro časovou diskretizaci použijeme linearizaci schématu v čase pomocí metody zamrzlých koeficientů a výslednou soustavu lineárních rovnic redukujeme pomocí metody hybridizace. V poslední části kapitoly stabilizujeme základní numerické schéma pomocí metody upwindu a popíšeme algoritmus výpočtu pro obě varianty numerického schématu.

Třetí kapitola poskytuje podrobný popis implementace numerického schématu bez upwindové stabilizace. V první části kapitoly popíšeme řešič využívající knihovnu `UMFPACK` pro řešení řídkých soustav lineárních rovnic pomocí přímé metody, ve druhé části se pak věnujeme implementaci pomocí

knihovny TNL a paralelizaci numerického schématu. V případě knihovny TNL jsou soustavy lineárních rovnic řešeny pomocí iteračních metod, které jsou součástí této knihovny.

V poslední kapitole jsou prezentovány výsledky numerických simulací. Konvergenci numerického schématu ověříme pomocí experimentální analýzy konvergence pro speciální případ se známým analytickým řešením. Dále porovnáme přesnost jednotlivých řešičů a obou variant numerického schématu. Na úloze se skutečnými materiálovými konstantami zkoumáme efektivitu TNL řešiče při použití různého hardware pro CPU a GPU.

Kapitola 1

Formulace problému

V první kapitole popíšeme fyzikální kontext a matematický model problematiky jednofázového proudění v porézním prostředí. Nejprve zavedeme základní pojmy potřebné pro popis porézního prostředí a jeho vlastností. Ve druhé části kapitoly popíšeme jednofázové proudění v porézním prostředí pomocí rovnice kontinuity, Darcyho zákona a stavové rovnice. V závěru kapitoly zformulujeme soustavu parciálních diferenciálních rovnic doplněnou o počáteční a okrajové podmínky, se kterou se budeme zabývat v dalších kapitolách.

Většina pojmů je převzata z [2] a [8], kde lze najít podrobnější popis této problematiky.

1.1 Porézní prostředí

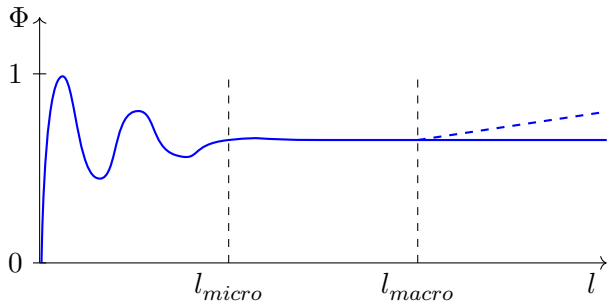
Porézním prostředím rozumíme takové prostředí, které je tvořené primárně pevnými látkami (*skeletem*), ale které obsahuje prázdné dutiny, neboli *póry*. Typickými příklady porézního prostředí jsou zemina složená ze štěrku, písku, jílu a organické hmoty, molitan, nebo rostlinná a živočišná tkáň. Póry 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 problém 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



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

objemu pórů v části uvažované oblasti vzhledem k celkovému objemu této části. Uvažujme oblast $\Omega \subset \mathbb{R}^d$ 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 Jednofázové proudění

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*.

[1] Representative Elementary Volume

1.2.1 Zákon zachování hmoty

Zákony zachování hrají důležitou roli při popisu zachovávajících se veličin. V případě jednofázového proudění lze zákon zachování hmoty vyjádřit pomocí *rovnice kontinuity* v diferenciálním tvaru

$$\frac{\partial(\Phi\rho)}{\partial t} + \nabla \cdot (\rho \boldsymbol{u}) = f, \quad \text{v } \Omega. \tag{1.3}$$

Veličiny vystupující v rovnici (1.3) mají následující význam:

$\Phi(\boldsymbol{x})$ *Porozita* porézního prostředí definovaná vztahem (1.2). Dále předpokládáme, že porozita je pouze funkcí polohy, což odpovídá případu, kdy skelet porézního prostředí je nedeformovatelný. Rovnici (1.3) pak lze zapsat ve tvaru

$$\Phi\frac{\partial\rho}{\partial t} + \nabla \cdot (\rho \boldsymbol{u}) = f, \quad \text{v } \Omega. \tag{1.4}$$

$\rho(\boldsymbol{x}, t)$ *Hustota* tekutiny v jednotkách $[\mathrm{kg} \cdot \mathrm{m}^{-3}]$. Obecně je hustota závislá na tlaku p, teplotě a složení dané fáze. V případě *nestlačitelných tekutin* je hustota konstantní, při aproximaci tekutiny ideálním plynem předpokládáme závislost $\rho = \rho(p)$ danou stavovou rovnicí ideálního plynu

$$\rho = \frac{pM}{RT}, \tag{1.5}$$

kde M je *molární hmotnost* $[\mathrm{kg} \cdot \mathrm{mol}^{-1}]$ uvažované tekutiny, R je *molární plynová konstanta* $[\mathrm{J} \cdot \mathrm{mol}^{-1} \cdot \mathrm{K}^{-1}]$ a T je *teplota* $[\mathrm{K}]$. V této práci se zabýváme pouze *izotermálním* prouděním probíhajícím při konstantní teplotě.

$\boldsymbol{u}(\boldsymbol{x}, t)$ *Měrný průtok*, neboli *Darcyho rychlost* $[\mathrm{m} \cdot \mathrm{s}^{-1}]$ udává množství tekutiny, která proteče jednotkovou plochou za 1 s. Souvislost se skutečnou průměrnou rychlostí $\boldsymbol{v}$ tekutiny v pórech je dána vztahem $\boldsymbol{v} = \frac{\boldsymbol{u}}{\Phi}$.

$f(\boldsymbol{x}, t)$ Člen charakterizující zdroje a propady uvnitř oblasti Ω, v jednotkách $[\mathrm{kg} \cdot \mathrm{m}^{-3} \cdot \mathrm{s}^{-1}]$.

1.2.2 Darcyho zákon

Darcyho zákon, který pro jednorozměrný případ empiricky odvodil roku 1856 Henri Darcy [4], představuje základní vztah pro makroskopický popis proudění v porézním prostředí. Obecný tvar lze odvodit z Navierových–Stokesových rovnic průměrováním přes REV nebo metodou homogenizace

$$\boldsymbol{u} = -\frac{1}{\mu}\mathbf{K}(\nabla p - \rho \boldsymbol{g}). \tag{1.6}$$

Nové veličiny vystupující v rovnici (1.6) mají následující význam:

$\mathbf{K}(\boldsymbol{x})$ *Propustnost* neboli *permeabilita* porézního prostředí $[\mathrm{m}^2]$. Podle vlastností $\mathbf{K}$ rozlišujeme několik typů prostředí. V obecném případě je $\mathbf{K}$ symetrický pozitivně definitní tenzor 2. řádu, přičemž hovoříme o *anizotropním prostředí*. Pokud je prostředí *izotropní*, lze místo tenzoru uvažovat skalár, tj. $\mathbf{K} = k\mathbf{I}$. Dále rozlišujeme *heterogenní prostředí*, kdy $\mathbf{K} = \mathbf{K}(\boldsymbol{x})$ je funkcí polohy, a *homogenní prostředí*, kdy $\mathbf{K}$ nezávisí na poloze.

μ *Dynamická viskozita* tekutiny $[\mathrm{Pa} \cdot \mathrm{s}]$. V této práci předpokládáme, že μ je konstantní.

$p(\boldsymbol{x}, t)$ *Tlak* tekutiny v jednotkách $[\mathrm{Pa}]$.

$\boldsymbol{g}$ Vektor *gravitačního zrychlení* v jednotkách $[\mathrm{m} \cdot \mathrm{s}^{-2}]$.

Darcyho zákon ve tvaru (1.6) je platný pouze pro pomalé proudění Newtonských tekutin v porézním prostředí s tuhým skeletem.[2] V této práci předpokládáme, že Darcyho zákon (1.6) platí ve všech uvažovaných případech.

1.2.3 Matematický model

Za výše popsaných předpokladů a při použití označení $\boldsymbol{q} = \rho \boldsymbol{u}$ lze rovnici kontinuity (1.4) a Darcyho zákon (1.6) přepsat ve tvaru

$$\Phi \frac{\partial \rho}{\partial t} + \nabla \cdot \boldsymbol{q} = f, \tag{1.7a}$$

$$\boldsymbol{q} = -\frac{\rho}{\mu} \mathbf{K} (\nabla p - \rho \boldsymbol{g}), \tag{1.7b}$$

kde tlak $p = p(\boldsymbol{x}, t)$ představuje neznámou funkci a závislost $\rho = \rho(p)$ hustoty fáze na tlaku je v případě stlačitelné plynné fáze dána stavovou rovnicí (1.5). Na oblasti $\Omega \times (0, T_f)$ předepíšeme počáteční a okrajové podmínky:

$$p(\boldsymbol{x}, 0) = p_0(\boldsymbol{x}), \qquad \forall \boldsymbol{x} \in \Omega, \tag{1.8a}$$

$$p(\boldsymbol{x}, t) = p_{Dir}(\boldsymbol{x}, t), \quad \forall (\boldsymbol{x}, t) \in \Gamma_{Dir} \times (0, T_f), \tag{1.8b}$$

$$\boldsymbol{n}(\boldsymbol{x}) \cdot \boldsymbol{q}(\boldsymbol{x}, t) = q_{Neu}(\boldsymbol{x}, t), \quad \forall (\boldsymbol{x}, t) \in \Gamma_{Neu} \times (0, T_f), \tag{1.8c}$$

kde $\boldsymbol{n}(\boldsymbol{x})$ označuje jednotkový vektor vnější normály v bodě $\boldsymbol{x} \in \partial\Omega$ a části hranice Γ_{Dir}, Γ_{Neu} splňují

$$\Gamma_{Dir} \cap \Gamma_{Neu} = \emptyset, \tag{1.9a}$$

$$\Gamma_{Dir} \cup \Gamma_{Neu} = \partial\Omega. \tag{1.9b}$$

Rovnice (1.7) představují soustavu parciálních diferenciálních rovnic doplněnou o algebraickou vazbu (1.5). Tuto soustavu lze dosazením rovnic (1.7b) a (1.5) do (1.7a) zredukovat na parciální diferenciální rovnici parabolického typu.

Kapitola 2

Numerický model

V této kapitole popíšeme odvození numerického schématu pro řešení soustavy parciálních diferenciálních rovnic (1.7) na oblasti $\Omega \subset \mathbb{R}^2$ s počátečními a okrajovými podmínkami (1.8). Použijeme při tom smíšenou formulaci metody konečných prvků (MFEM[1]) pro aproximaci funkcí p a $\boldsymbol{q}$. Standardní formulace této metody však vede na soustavu s indefinitní maticí [9, 3], proto použijeme hybridní smíšenou metodu konečných prvků (MHFEM[2]), která vede na soustavu s pozitivně definitní maticí.

V jednotlivých podkapitolách nejprve diskretizujeme prostorovou oblast Ω pomocí numerické sítě, následně diskretizujeme soustavu (1.7) pomocí metod konečných prvků a konečných objemů (FVM[3]) a schéma získané časovou diskretizací pomocí Eulerovy metody linearizujeme pomocí metody zamrzlých koeficientů. Výslednou soustavu lineárních rovnic redukujeme pomocí metody hybridizace. Na závěr popíšeme modifikaci základního numerického schématu pomocí upwindové stabilizace a algoritmus výpočtu numerického řešení pro obě varianty numerického schématu.

Při formulaci numerického schématu vycházíme především z [3] a [8].

2.1 Diskretizace prostorové oblasti

Společným základem pro metody konečných prvků a konečných objemů je diskretizace prostorové oblasti pomocí numerické sítě. Oblast $\Omega \subset \mathbb{R}^d$ pokryjeme konečným systémem $\mathcal{K}_h$ navzájem disjunktních oblastí K s jednoduchou geometrickou strukturu,

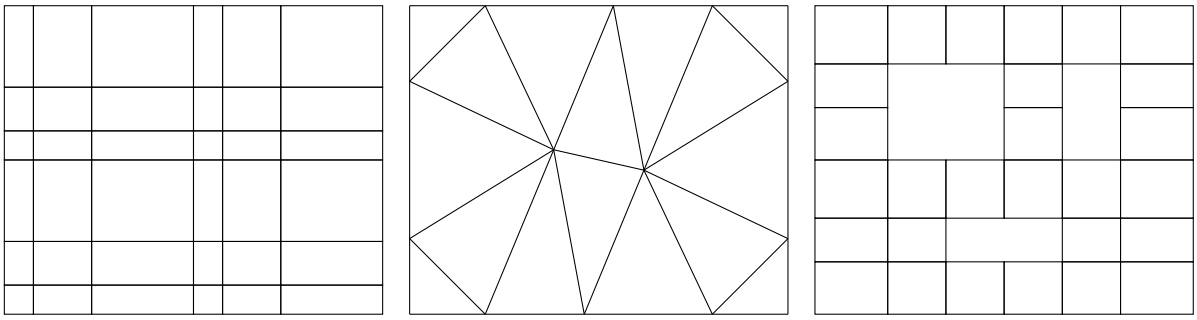
$$\bigcup_{K \in \mathcal{K}_h} \overline{K} = \overline{\Omega}. \tag{2.1}$$

Prvky systému $\mathcal{K}_h$ nazýváme *elementy* a $h > 0$ označuje průměr největšího elementu $K \in \mathcal{K}_h$. Nejčastěji používané geometrie elementů jsou:

- v $\mathbb{R}^1$: úsečky,
- v $\mathbb{R}^2$: trojúhelníky a čtyřúhelníky,
- v $\mathbb{R}^3$: čtyřstěny, kvádry, jehlany,

přičemž se pro jednoduchost zpravidla předpokládá, že oblast Ω má takový tvar, že ji lze přesně pokrýt elementy požadovaných tvarů.

[1] Mixed Finite Element Method
[2] Mixed Hybrid Finite Element Method
[3] Finite Volume Method



(a) Strukturovaná síť (b) Nestrukturovaná síť (c) Nekonformní síť

Obrázek 2.1: Příklady numerických sítí pokrývajících obdélníkovou oblast. Na obrázku (a) je zobrazena strukturovaná konformní síť, na obrázku (b) nestrukturovaná konformní síť a na obrázku (c) nekonformní síť.

Hranice ∂K elementu $K \in \mathcal{K}_h$ je tvořena konečným sjednocením *vrcholů*, *hran* a *stěn*. Množiny všech vrcholů, hran a stěn označíme po řadě $\mathcal{V}_h$, $\mathcal{E}_h$ a $\mathcal{F}_h$. *Sítí* $\mathcal{M}_h$ pak nazýváme množinu všech vrcholů, hran, stěn a elementů, tedy $\mathcal{M}_h = \mathcal{V}_h \cup \mathcal{E}_h \cup \mathcal{F}_h \cup \mathcal{K}_h$. Pro sítě v $\mathbb{R}^2$ dále zavedeme značení, které využijeme v následujících kapitolách. Počty všech vrcholů, hran a elementů sítě $\mathcal{M}_h$ označíme po řadě N_V, N_E a N_K. Množinu vnějších hran sítě $\mathcal{M}_h$ označíme jako $\mathcal{E}_h^{ext} = \{E \in \mathcal{E}_h; E \subset \Gamma\}$ a analogicky definujeme množinu vnitřních hran $\mathcal{E}_h^{int} = \{E \in \mathcal{E}_h; E \not\subset \Gamma\}$. Pro množinu hran příslušných danému elementu zavedeme označení $\mathcal{E}_K = \{E \in \mathcal{E}_h; E \subset \partial K\}$.

Pokud mají elementy společnou část hranice, říkáme, že jsou *sousední*. Síť, ve které pro každé dva sousední elementy $K_1, K_2 \in \mathcal{K}_h$ platí $\partial K_1 \cap \partial K_2 \in \mathcal{M}_h$, nazýváme *konformní*. Naopak *nekonformní* sítě jsou charakterizovány tím, že společná část hranice sousedních elementů může tvořit pouze část nějaké hrany nebo stěny. Dále rozlišujeme *strukturované* sítě, které jsou tvořeny elementy pravidelných tvarů, např. obdélníky. Pokud mají všechny elementy strukturované sítě stejné rozměry, jedná se o *ekvidistantní* síť. Elementy *nestrukturovaných* sítí mají různé tvary a rozměry nebo jsou rozmístěny nepravidelně. Příklady různých typů sítí jsou zobrazeny na obr. 2.1.

V praxi se používají různé typy sítí, volba sítě závisí na typu studovaného problému a použitém numerickém schématu. V této práci se dále omezíme na ekvidistantní konformní obdélníkové sítě a symboly h_x, resp. h_y označíme rozměry elementů v x-ovém, resp. y-ovém směru.

2.2 Diskretizace Darcyho zákona

2.2.1 Slabá formulace

Prvním krokem při odvození metody konečných prvků je zeslabení pojmu řešení parciální diferenciální rovnice (1.7) spolu s podmínkami (1.8) a vazbou (1.5). Předpokládáme, že $\rho \geq \rho_0 > 0$ na Ω a že tenzor permeability $\mathbf{K}$ je symetrický a pozitivně definitní, a tedy regulární [1]. Rovnici (1.7b) lze tedy přepsat ve tvaru

$$\nabla p = -\frac{\mu}{\rho}\mathbf{K}^{-1}\boldsymbol{q} + \rho\boldsymbol{g}. \tag{2.2}$$

Rovnici (2.2) dále vynásobíme libovolnou testovací funkcí $\boldsymbol{\omega} \in \left[C^\infty(\overline{\Omega})\right]^d$ a integrujeme přes $\Omega \subset \mathbb{R}^d$,

$$\int_\Omega \nabla p \cdot \boldsymbol{\omega} = -\int_\Omega \frac{\mu}{\rho}(\mathbf{K}^{-1}\boldsymbol{q}) \cdot \boldsymbol{\omega} + \int_\Omega \rho \boldsymbol{g} \cdot \boldsymbol{\omega}. \tag{2.3}$$

Úpravou integrálu na levé straně rovnice (2.3) pomocí Greenovy formule dále dostaneme

$$\int_\Gamma p\boldsymbol{\omega} \cdot \boldsymbol{n} - \int_\Omega p\nabla \cdot \boldsymbol{\omega} = -\int_\Omega \frac{\mu}{\rho}(\mathbf{K}^{-1}\boldsymbol{q}) \cdot \boldsymbol{\omega} + \int_\Omega \rho \boldsymbol{g} \cdot \boldsymbol{\omega}, \tag{2.4}$$

kde $\boldsymbol{n}$ značí jednotkový vektor vnější normály oblasti Ω na hranici Γ. Dále se omezíme na funkce $\boldsymbol{\omega} \in \left[C^\infty(\overline{\Omega})\right]^d$, pro které platí $\boldsymbol{\omega} \cdot \boldsymbol{n} = 0$ na Γ_{Neu} a pro úpravu prvního integrálu na levé straně rovnice (2.4) využijeme okrajové podmínky (1.8b):

$$\int_{\Gamma_{Dir}} p_{Dir}\boldsymbol{\omega} \cdot \boldsymbol{n} - \int_\Omega p\nabla \cdot \boldsymbol{\omega} = -\int_\Omega \frac{\mu}{\rho}(\mathbf{K}^{-1}\boldsymbol{q}) \cdot \boldsymbol{\omega} + \int_\Omega \rho \boldsymbol{g} \cdot \boldsymbol{\omega}. \tag{2.5}$$

Zřejmě každé řešení $(p, \boldsymbol{q})$ původní rovnice (1.7) na oblasti Ω řeší také (2.5) pro libovolnou volbu funkce $\boldsymbol{\omega} \in \left[C^\infty(\overline{\Omega})\right]^d$, rovnici (2.5) však lze splnit za slabších předpokladů než jsou požadavky na hladkost klasického řešení.

Na $\Omega \subset \mathbb{R}^d$ definujeme prostor funkcí $\mathbf{H}(\mathrm{div}, \Omega)$,

$$\mathbf{H}(\mathrm{div}, \Omega) = \left\{\boldsymbol{\omega} \in [L^2(\Omega)]^d \mid \nabla \cdot \boldsymbol{\omega} \in L^2(\Omega)\right\}. \tag{2.6}$$

Pro funkce $\boldsymbol{\omega} \in \mathbf{H}(\mathrm{div}, \Omega)$ lze definovat normálovou stopu $\boldsymbol{\omega} \cdot \boldsymbol{n}|_\Gamma$, která splňuje Greenovu formuli

$$\int_\Omega \nabla \cdot \boldsymbol{\omega} v + \int_\Omega \boldsymbol{\omega} \cdot \nabla v = \int_\Gamma \boldsymbol{\omega} \cdot \boldsymbol{n} v, \qquad \forall v \in H^1(\Omega). \tag{2.7}$$

kde $H^1(\Omega)$ označuje Sobolevův prostor [3]. Pro $\Upsilon \subset \Gamma$ tedy můžeme definovat podprostor prostoru $\mathbf{H}(\mathrm{div}, \Omega)$ obsahující funkce s nulovou normálovou stopou na Υ,

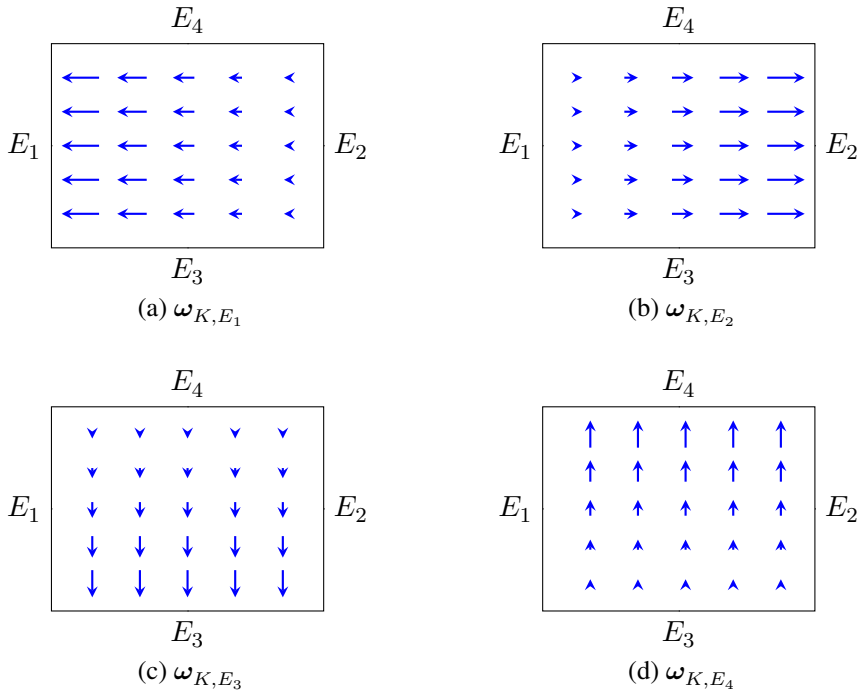
$$\mathbf{H}_{0,\Upsilon}(\mathrm{div}, \Omega) = \{\boldsymbol{\omega} \in \mathbf{H}(\mathrm{div}, \Omega) \mid \boldsymbol{\omega} \cdot \boldsymbol{n}|_\Upsilon = 0\}. \tag{2.8}$$

Pro splnění Neumannovy okrajové podmínky (1.8c) dále předpokládáme, že existuje funkce $\widehat{\boldsymbol{q}} \in \mathbf{H}(\mathrm{div}, \Omega)$ taková, že $\widehat{\boldsymbol{q}} \cdot \boldsymbol{n}|_{\Gamma_{Neu}} \equiv q_{Neu}$. Funkce $p \in L^2(\Omega)$ a $\boldsymbol{q} \in \widehat{\boldsymbol{q}} + \mathbf{H}_{0,\Gamma_{Neu}}(\mathrm{div}, \Omega)$ splňující rovnici (2.5) pro každou $\boldsymbol{\omega} \in \mathbf{H}_{0,\Gamma_{Neu}}(\mathrm{div}, \Omega)$ nazýváme *slabým řešením* rovnice (2.2).

2.2.2 Raviartův–Thomasův prostor

Pro účely diskretizace funkce $\boldsymbol{q}$ si připravíme vhodný konečnědimenzionální podprostor prostoru $\mathbf{H}(\mathrm{div}, \Omega)$. Jedny z používaných prostorů jsou Raviartovy–Thomasovy prostory, v této práci použijeme Raviartův–Thomasův prostor nejnižšího řádu $\mathbf{RT}_0(\mathcal{K}_h)$.

Při definici prostoru $\mathbf{RT}_0(\mathcal{K}_h)$ využijeme rozklad oblasti Ω na konečný systém podoblastí $K \in \mathcal{K}_h$, kde na každém takovém elementu K definujeme prostor $\mathbf{RT}_0(K)$ jako konečnědimenzionální prostor generovaný bazickými funkcemi $\boldsymbol{\omega}_{K,E} \in \mathbf{H}(\mathrm{div}, K)$, $E \in \mathcal{E}_K$. Konkrétní volba funkcí $\boldsymbol{\omega}_{K,E}$ je závislá



(a) $\boldsymbol{\omega}_{K,E_1}$ (b) $\boldsymbol{\omega}_{K,E_2}$

(c) $\boldsymbol{\omega}_{K,E_3}$ (d) $\boldsymbol{\omega}_{K,E_4}$

Obrázek 2.2: Vizualizace bazických funkcí prostoru $\mathbf{RT}_0(K)$ na obdélníkovém elementu K.

na geometrii elementů K. V $\mathbb{R}^2$ pro obdélníkový element $K = (0, h_x) \times (0, h_y)$ volíme bazické funkce ve tvaru

$$\begin{aligned} \boldsymbol{\omega}_{K,E_1}(\boldsymbol{x}) &= \frac{1}{|K|}\begin{pmatrix} x_1 - h_x \\ 0 \end{pmatrix}, & \boldsymbol{\omega}_{K,E_2}(\boldsymbol{x}) &= \frac{1}{|K|}\begin{pmatrix} x_1 \\ 0 \end{pmatrix}, \\ \boldsymbol{\omega}_{K,E_3}(\boldsymbol{x}) &= \frac{1}{|K|}\begin{pmatrix} 0 \\ x_2 - h_y \end{pmatrix}, & \boldsymbol{\omega}_{K,E_4}(\boldsymbol{x}) &= \frac{1}{|K|}\begin{pmatrix} 0 \\ x_2 \end{pmatrix}, \end{aligned} \tag{2.9}$$

kde x_1, x_2 značí složky vektoru $\boldsymbol{x} \in \mathbb{R}^2$, $|K|$ označuje objem elementu K a E_1, E_2, E_3, E_4 značí postupně levou, pravou, dolní a horní hranu elementu K. Bazické funkce jsou voleny tak, aby splňovaly podmínky

$$\nabla \cdot \boldsymbol{\omega}_{K,E}(\boldsymbol{x}) = \frac{1}{|K|}, \qquad \forall \boldsymbol{x} \in K, \tag{2.10a}$$

$$\boldsymbol{\omega}_{K,E}(\boldsymbol{x}) \cdot \boldsymbol{n}_{K,F} = \frac{1}{|F|}\delta_{EF}, \qquad \forall \boldsymbol{x} \in F, \tag{2.10b}$$

kde $\boldsymbol{n}_{K,F}$ označuje jednotkový vektor vnější normály na hraně $F \in \mathcal{E}_K$ vzhledem k elementu K, $|F|$ označuje délku hrany F a δ_{EF} označuje Kroneckerovo delta. Z podmínek (2.10) dále plyne

$$\int_K \nabla \cdot \boldsymbol{\omega}_{K,E}(\boldsymbol{x}) = 1, \tag{2.11a}$$

$$\int_F \omega_{K,E}(\boldsymbol{x}) \cdot \boldsymbol{n}_{K,F} = \delta_{EF}. \tag{2.11b}$$

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

$$\mathbf{RT}_0^0(\mathcal{K}_h) = \left\{ \boldsymbol{\omega} \in [L^2(\Omega)]^d \;\middle|\; \forall K \in \mathcal{K}_h,\ \boldsymbol{\omega}|_K \in \mathbf{H}(\mathrm{div}, K) \right\}. \tag{2.12}$$

Prostor $\mathbf{RT}_0^0(\mathcal{K}_h)$ však není podprostorem prostoru $\mathbf{H}(\mathrm{div},\Omega)$. Ve smyslu tvrzení 1.2 v knize [3, str. 95] navíc po funkcích $\boldsymbol{\omega} \in \mathbf{RT}_0^0(\mathcal{K}_h)$ požadujeme spojitost normálových stop na vnitřních hranách sítě. Tuto vlastnost můžeme ekvivalentně vyjádřit pomocí bilanční podmínky, kterou použijeme pro definici prostoru $\mathbf{RT}_0(\mathcal{K}_h)$:

$$\mathbf{RT}_0(\mathcal{K}_h) = \left\{ \boldsymbol{\omega} \in [L^2(\Omega)]^d \;\middle|\; \forall K \in \mathcal{K}_h,\ \boldsymbol{\omega}|_K \in \mathbf{RT}_0(K);\ \forall E \in \mathcal{E}_h^{int}, E \in \mathcal{E}_{K_1} \cap \mathcal{E}_{K_2}, \int_E \boldsymbol{\omega}|_{K_1} \cdot \boldsymbol{n}_{K_1,E} + \int_E \boldsymbol{\omega}|_{K_2} \cdot \boldsymbol{n}_{K_2,E} = 0 \right\}. \quad (2.13)$$

2.2.3 Aproximace neznámých funkcí

V rovnici (2.5) aproximujeme funkce $p \in L^2(\Omega)$ a $\boldsymbol{q} \in \widehat{\boldsymbol{q}} + \mathbf{H}_{0,\Gamma_{Neu}}(\mathrm{div},\Omega)$ pomocí funkcí $p_h \in V_h$ a $\boldsymbol{q}_h \in \widehat{\boldsymbol{q}}_h + \mathbf{W}_h$, kde $V_h \subset L^2(\Omega)$ a $\mathbf{W}_h \subset \mathbf{H}_{0,\Gamma_{Neu}}(\mathrm{div},\Omega)$ jsou zvolené konečnědimenzionální podprostory a $\widehat{\boldsymbol{q}}_h$ aproximuje funkci $\widehat{\boldsymbol{q}}$ na Γ_{Neu}. Prostor V_h volíme jako prostor funkcí konstantních na elementech $K \in \mathcal{K}_h$, tedy

$$V_h = \mathrm{span}\left\{\chi_K \mid K \in \mathcal{K}_h\right\}, \quad (2.14)$$

kde χ_K označuje charakteristickou funkci elementu $K \subset \Omega$. Vyjádřením funkce p_h pomocí bazických funkcí prostoru V_h získáme

$$p_h = \sum_{K \in \mathcal{K}_h} p_K \chi_K. \quad (2.15)$$

Aproximaci p funkcí p_h rozumíme ve smyslu ortogonální projekce funkce p do konečnědimenzionálního podprostoru V_h, tedy

$$\int_\Omega p\chi_K \approx \int_\Omega p_h\chi_K = \int_K \sum_{K' \in \mathcal{K}_h} p_{K'}\chi_{K'} = p_K \int_K \chi_K = p_K|K|. \quad (2.16)$$

Koeficienty p_K tedy odpovídají středním hodnotám funkce p na jednotlivých elementech $K \in \mathcal{K}_h$.

Prostor $\mathbf{W}_h$ volíme jako podprostor Raviartova–Thomasova prostoru $\mathbf{RT}_0(\mathcal{K}_h)$ s ohledem na okrajové podmínky,

$$\mathbf{W}_h = \mathbf{RT}_0^0(\mathcal{K}_h) \cap \mathbf{H}_{0,\Gamma_{Neu}}(\mathrm{div},\Omega). \quad (2.17)$$

Nejprve využijeme reprezentace funkce $\boldsymbol{q}_h$ na každém elementu $K \in \mathcal{K}_h$ pomocí bazických funkcí $\boldsymbol{\omega}_{K,E}$ prostoru $\mathbf{RT}_0(K)$,

$$\boldsymbol{q}_h|_K = \sum_{E \in \mathcal{E}_K} q_{K,E}\boldsymbol{\omega}_{K,E}. \quad (2.18)$$

Podobně jako v případě skalární funkce, aproximaci vektorové funkce $\boldsymbol{q}$ funkcí $\boldsymbol{q}_h$ rozumíme ve smyslu ortogonální projekce funkce $\boldsymbol{q} - \widehat{\boldsymbol{q}} \in \mathbf{H}_{0,\Gamma_{Neu}}(\mathrm{div},\Omega)$ do podprostoru $\mathbf{W}_h$. Integrací normálové stopy $\boldsymbol{q}|_K \cdot \boldsymbol{n}_{K,E}$ vzhledem k elementu K přes hranu E dostaneme

$$\int_E \boldsymbol{q}|_K \cdot \boldsymbol{n}_{K,E} \approx \int_E \boldsymbol{q}_h|_K \cdot \boldsymbol{n}_{K,E} = \int_E \sum_{F \in \mathcal{E}_K} q_{K,F}\boldsymbol{\omega}_{K,F} \cdot \boldsymbol{n}_{K,E} = \int_E \frac{1}{|E|} q_{K,E} = q_{K,E}. \quad (2.19)$$

Koeficienty $q_{K,E}$ tedy odpovídají numerickému toku vektorové funkce $\boldsymbol{q}$ hranou $E \in \mathcal{E}_K$ ve směru vnější normály vzhledem k elementu K. Dále uvažujme dva sousední elementy $K_1, K_2 \in \mathcal{K}_h$ se společnou hranou $E \in \mathcal{E}_{K_1} \cap \mathcal{E}_{K_2}$. Podmínku spojitosti normálových stop funkce $\boldsymbol{q}_h$ na hraně E, požadovanou v definici prostoru $\mathbf{RT}_0(\mathcal{K}_h)$ (2.13), nyní můžeme vyjádřit pomocí koeficientů $q_{K_1,E}$ a $q_{K_2,E}$ ve tvaru

$$q_{K_1,E} + q_{K_2,E} = 0. \tag{2.20}$$

Tato podmínka odpovídá předpokladu nulových zdrojových členů na hraně E a představuje tedy lokální bilanci hmoty.

V rovnici (2.5) nahradíme funkce p a $\boldsymbol{q}$ funkcemi p_h a $\boldsymbol{q}_h$, přičemž testovací funkce $\boldsymbol{\omega}$ bereme z prostoru $\mathbf{W}_h$:

$$\int_{\Gamma_{Dir}} p_{Dir}\boldsymbol{\omega}\cdot\boldsymbol{n} - \int_\Omega p_h \nabla\cdot\boldsymbol{\omega} = -\int_\Omega \frac{\mu}{\rho_h}(\mathbf{K}^{-1}\boldsymbol{q}_h)\cdot\boldsymbol{\omega} + \int_\Omega \rho_h \boldsymbol{g}\cdot\boldsymbol{\omega}, \qquad \forall\boldsymbol{\omega}\in\mathbf{W}_h, \tag{2.21}$$

kde $\rho_h = \rho(p_h)$. V hybridní formulaci dále přejdeme k prostorům $\mathbf{RT}_0(K)$ definovaných na každém elementu $K \in \mathcal{K}_h$. Integrály přes oblast Ω můžeme zapsat jako součty integrálů přes jednotlivé elementy a na každém elementu K vyjádřit $\boldsymbol{\omega}|_K$ pomocí bazických funkcí prostoru $\mathbf{RT}_0(K)$.

Zabývejme se nyní integrálem přes Γ_{Dir} v rovnici (2.21). Pomocí Greenovy formule (2.7) můžeme snadno nahlédnout, že pro libovolné funkce $v \in H^1(\Omega)$ a $\boldsymbol{\omega} \in \mathbf{H}(\mathrm{div},\Omega)$ platí rovnost

$$\int_\Gamma v\boldsymbol{\omega}\cdot\boldsymbol{n} = \sum_{K\in\mathcal{K}_h}\int_{\partial K} v\boldsymbol{\omega}|_K\cdot\boldsymbol{n}_{\partial K}, \tag{2.22}$$

kde $\boldsymbol{n}_{\partial K}$ označuje vektor vnější normály vzhledem k elementu K na hranici ∂K. Pro funkci $p \in V_h$ však kvůli nespojitostem mezi sousedními elementy nelze dobře definovat stopu na hranici elementu ∂K, pomůžeme si tedy trikem popsaným v knize [3, kap. 5.1.2]. Definujme prostor funkcí po částech konstantních na hranách sítě $\mathcal{E}_h$,

$$\Lambda = V_h(\mathcal{E}_h), \tag{2.23}$$

a jeho podprostor

$$\Lambda_{Dir} = \left\{ \lambda_h \in \Lambda \;\middle|\; \forall E \in \mathcal{E}_h \cap \Gamma_{Dir},\ \int_E \lambda_h = \int_E p_{Dir} \right\}. \tag{2.24}$$

Buď dále $(p_h, \boldsymbol{q}_h)$ řešení smíšené formulace (2.21). Pak existuje právě jedna funkce $\widehat{p}_h \in \Lambda_{Dir}$ taková, že platí

$$\sum_{K\in\mathcal{K}_h}\int_{\partial K} \widehat{p}_h\boldsymbol{\omega}\cdot\boldsymbol{n}_{\partial K} - \int_\Omega p_h\nabla\cdot\boldsymbol{\omega} = -\int_\Omega \frac{\mu}{\rho_h}(\mathbf{K}^{-1}\boldsymbol{q}_h)\cdot\boldsymbol{\omega} + \int_\Omega \rho_h\boldsymbol{g}\cdot\boldsymbol{\omega}, \qquad \forall\boldsymbol{\omega}\in\mathbf{RT}_0^0(\mathcal{K}_h). \tag{2.25}$$

Důležitým výsledkem je, že v rovnici (2.25) nepožadujeme spojitost funkcí $\boldsymbol{\omega}$ na vnitřních hranách sítě. Funkci $\boldsymbol{q}_h$ však stále hledáme v prostoru $\mathbf{RT}_0(\mathcal{K}_h)$, což zajistíme pomocí propojovacích podmínek (2.19).

2.2.4 Diskrétní Darcyho zákon

Rovnice (2.25) je ekvivalentní soustavě lokálních rovnic pro jednotlivé elementy:

$$\int_{\partial K} \widehat{p}_h \boldsymbol{\omega}_{K,E} \cdot \boldsymbol{n}_{\partial K} - \int_K p_h \nabla \cdot \boldsymbol{\omega}_{K,E} = -\int_K \frac{\mu}{\rho_h} (\mathbf{K}^{-1} \boldsymbol{q}_h) \cdot \boldsymbol{\omega}_{K,E} + \int_K \rho_h \boldsymbol{g} \cdot \boldsymbol{\omega}_{K,E},$$
$$\forall K \in \mathcal{K}_h,\ \forall \boldsymbol{\omega}_{K,E} \in \mathbf{RT}_0(K). \tag{2.26}$$

S využitím vlastností bazických funkcí prostoru $\mathbf{RT}_0(K)$ (2.11) upravíme levou stranu rovnice (2.26) do tvaru

$$\begin{aligned}\int_{\partial K} \widehat{p}_h \boldsymbol{\omega}_{K,E} \cdot \boldsymbol{n}_{\partial K} - \int_K p_h \nabla \cdot \boldsymbol{\omega}_{K,E} &= \sum_{F \in \mathcal{E}_K} \int_F \widehat{p}_h \boldsymbol{\omega}_{K,E} \cdot \boldsymbol{n}_{K,F} - \int_K p_h \nabla \cdot \boldsymbol{\omega}_{K,E} \\ &= \sum_{F \in \mathcal{E}_K} p_F \int_F \boldsymbol{\omega}_{K,E} \cdot \boldsymbol{n}_{K,F} - p_K \int_K \nabla \cdot \boldsymbol{\omega}_{K,E} \\ &= p_E - p_K, \end{aligned} \tag{2.27}$$

přičemž p_E představuje stopu tlaku p na hraně E, $p_E \equiv \widehat{p}_h|_E$. První integrál na pravé straně rovnice (2.26) zapíšeme ve tvaru

$$\int_K \frac{\mu}{\rho_h} (\mathbf{K}^{-1} \boldsymbol{q}_h) \cdot \boldsymbol{\omega}_{K,E} = \sum_{F \in \mathcal{E}_K} q_{K,F} \frac{\mu}{\rho_K} \int_K \boldsymbol{\omega}_{K,E} \cdot (\mathbf{K}^{-1} \boldsymbol{\omega}_{K,F}), \tag{2.28}$$

kde $\rho_K \equiv \rho(p_K)$. Celkově pomocí (2.27) a (2.28) přepíšeme rovnici (2.26) ve tvaru

$$p_E - p_K = -\sum_{F \in \mathcal{E}_K} q_{K,F} \widetilde{a}_{K,E,F} + \widetilde{G}_{K,E}, \tag{2.29}$$

přičemž pro integrály na pravé straně rovnice (2.26) jsme zavedli označení

$$a_{K,E,F} = \mu \int_K \boldsymbol{\omega}_{K,E} \cdot (\mathbf{K}^{-1} \boldsymbol{\omega}_{K,F}), \qquad \widetilde{a}_{K,E,F} = \frac{1}{\rho_K} a_{K,E,F}, \tag{2.30a}$$

$$G_{K,E} = \int_K \boldsymbol{g} \cdot \boldsymbol{\omega}_{K,E}, \qquad \widetilde{G}_{K,E} = \rho_K G_{K,E}. \tag{2.30b}$$

Značení (2.30) lze lokálně na každém elementu $K \in \mathcal{K}_h$ zapsat v maticovém resp. vektorovém tvaru

$$\mathbf{a}_K = \left[a_{K,E,F}\right]_{E,F \in \mathcal{E}_K}, \qquad \widetilde{\mathbf{a}}_K = \left[\widetilde{a}_{K,E,F}\right]_{E,F \in \mathcal{E}_K} = \frac{1}{\rho_K} \mathbf{a}_K, \tag{2.31a}$$

$$\boldsymbol{G}_K = \left[G_{K,E}\right]_{E \in \mathcal{E}_K}, \qquad \widetilde{\boldsymbol{G}}_K = \left[\widetilde{G}_{K,E}\right]_{E \in \mathcal{E}_K} = \rho_K \boldsymbol{G}_{K,E}. \tag{2.31b}$$

Za předpokladu, že $\mathbf{K}$ je symetrický a pozitivně definitní tenzor, je matice $\mathbf{a}_K$ symetrická a pozitivně definitní, a tedy také invertibilní [1]. Označme tedy

$$\mathbf{a}_K^{-1} = \mathbf{A}_K = \left[A_{K,E,F}\right]_{E,F \in \mathcal{E}_K}, \tag{2.32}$$

$$\widetilde{\mathbf{a}}_K^{-1} = \widetilde{\mathbf{A}}_K = \left[\widetilde{A}_{K,E,F}\right]_{E,F \in \mathcal{E}_K} = \rho_K \mathbf{A}_{K,E,F}. \tag{2.33}$$

Inverzí (2.29) získáme vztah

$$q_{K,E} = \sum_{F \in \mathcal{E}_K} \widetilde{A}_{K,E,F} \left(p_K - p_F + \widetilde{G}_{K,F}\right), \qquad \forall K \in \mathcal{K}_h,\ \forall E \in \mathcal{E}_K, \tag{2.34}$$

který představuje Darcyho zákon v diskrétní podobě.

2.2.5 Výpočet prvků lokální matice

Dále budeme potřebovat znát hodnoty prvků lokálních matic $\mathbf{A}_K$ a vektorů $\boldsymbol{G}_K$ definovaných na každém elementu $K \in \mathcal{K}_h$. Uvažujme tedy referenční element $K = (0, h_x) \times (0, h_y)$ o objemu $|K| = h_x h_y$, přičemž symboly E_1, E_2, E_3, E_4 stejně jako v sekci 2.2.2 označíme postupně levou, pravou, dolní a horní hranu elementu K.

Dále předpokládejme, že gravitační zrychlení působí pouze ve směru osy y a označme $\boldsymbol{g} = (0, -g_y)$. Podle definice bazických funkcí (2.9) platí $\boldsymbol{g} \cdot \boldsymbol{\omega}_{K,E_1} = \boldsymbol{g} \cdot \boldsymbol{\omega}_{K,E_2} = 0$ a tedy také $G_{K,E_1} = G_{K,E_2} = 0$. Výpočtem integrálu (2.30b) určíme zbývající složky $G_{K,E_3} = -G_{K,E_4} = \frac{1}{2} g_y h_y$. Celkově lze vektor $\boldsymbol{G}_K$ zapsat ve tvaru

$$\boldsymbol{G}_K = \frac{1}{2} g_y h_y \begin{pmatrix} 0 \\ 0 \\ 1 \\ -1 \end{pmatrix}. \tag{2.35}$$

Prvky matice $\mathbf{a}_K$ určíme pro případ, kdy tenzor $\mathbf{K} = k\mathbf{I}$ popisuje izotropní prostředí a je konstantní na elementu K. Při výpočtu využijeme toho, že skalární součin $\boldsymbol{\omega}_{K,E} \cdot \boldsymbol{\omega}_{K,F}$ bazických funkcí příslušných sousedním hranám $E, F \in \mathcal{E}_K$ je identicky roven nule na K. Navíc víme, že matice $\mathbf{a}_K$ je symetrická, takže výpočtem integrálu (2.30a) stačí určit hodnoty čtyř diagonálních a dvou mimodiagonálních prvků. Celkově lze matici $\mathbf{a}_K$ zapsat ve tvaru

$$\mathbf{a}_K = \frac{1}{6} \mu k^{-1} \begin{pmatrix} 2\frac{h_x}{h_y} & -\frac{h_x}{h_y} & 0 & 0 \\ -\frac{h_x}{h_y} & 2\frac{h_x}{h_y} & 0 & 0 \\ 0 & 0 & 2\frac{h_y}{h_x} & -\frac{h_y}{h_x} \\ 0 & 0 & -\frac{h_y}{h_x} & 2\frac{h_y}{h_x} \end{pmatrix}. \tag{2.36}$$

V článku [17] je ukázáno, že výsledná matice soustavy pro MHFEM není při použití obdélníkových elementů M-matice, což může způsobit oscilace v numerickém řešení. Proto použijeme metodu *mass-lumping* a integrály (2.30a) spočítáme pouze přibližně pomocí numerické integrace

$$\int_K \xi(\boldsymbol{x}) \mathrm{d}\boldsymbol{x} \approx \frac{|K|}{4} \sum_{i=1}^{4} \xi(\boldsymbol{x}_i), \tag{2.37}$$

kde body $\boldsymbol{x}_i$ označují vrcholy elementu K. Matici $\mathbf{a}_K$ tedy aproximujeme maticí $\mathbf{a}_K^{(\ell)}$, jejíž prvky jsou dány integrály (2.30a) spočítanými pomocí (2.37). Matice $\mathbf{a}_K^{(\ell)}$ je diagonální a má tvar

$$\mathbf{a}_K \approx \mathbf{a}_K^{(\ell)} = \frac{1}{2} \mu k^{-1} \begin{pmatrix} \frac{h_x}{h_y} & 0 & 0 & 0 \\ 0 & \frac{h_x}{h_y} & 0 & 0 \\ 0 & 0 & \frac{h_y}{h_x} & 0 \\ 0 & 0 & 0 & \frac{h_y}{h_x} \end{pmatrix}. \tag{2.38}$$

Matici $\mathbf{A}_K = \mathbf{a}_K^{-1}$ pak aproximujeme inverzí matice $\mathbf{a}_K^{(\ell)}$, tedy

$$\mathbf{A}_K \approx 2\frac{k}{\mu} \begin{pmatrix} \frac{h_y}{h_x} & 0 & 0 & 0 \\ 0 & \frac{h_y}{h_x} & 0 & 0 \\ 0 & 0 & \frac{h_x}{h_y} & 0 \\ 0 & 0 & 0 & \frac{h_x}{h_y} \end{pmatrix}. \tag{2.39}$$

2.3 Diskretizace rovnice kontinuity

Rovnici (1.7a) diskretizujeme pomocí metody konečných objemů (CFVM[4]) s centrem konečného objemu v těžišti elementu.[10] Řešení této rovnice hledáme na oblasti $\Omega \times (0, T_f)$, přičemž pro diskretizaci prostorové oblasti Ω použijeme stejnou síť $\mathcal{K}_h$, jakou jsme použili v kapitole 2.2. Časový interval $[0, T_f]$ diskretizujeme pomocí množiny $\mathcal{T}$,

$$\mathcal{T} = \left\{0 = t_0 < t_1 < ... < t_n = T_f\right\}, \tag{2.40}$$

přičemž délky časových kroků označíme jako $\Delta t_i = t_{i+1} - t_i$. Dále budeme značit ξ^i hodnotu funkce ξ v čase $t_i \in \mathcal{T}$, tedy $\xi^i = \xi(\cdot, t_i)$.

Dosazením (1.5) do rovnice (1.7a) a integrací přes libovolný element $K \in \mathcal{K}_h$ dostaneme

$$\frac{M}{RT}\frac{\mathrm{d}}{\mathrm{d}t}\int_K \Phi(\boldsymbol{x})p(\boldsymbol{x},t) + \int_K \nabla \cdot \boldsymbol{q}(\boldsymbol{x},t) = \int_K f(\boldsymbol{x},t). \tag{2.41}$$

Dále předpokládáme, že porozita Φ je konstantní na elementu K. První integrál na levé straně a integrál na pravé straně rovnice (2.41) nahradíme středními hodnotami příslušných veličin, funkce p a f tedy aproximujeme pomocí funkčního prostoru V_h definovaného v kapitole 2.2.3. Druhý integrál na levé straně rovnice (2.41) upravíme pomocí Greenovy formule, čímž dostaneme

$$\frac{\Phi_K M}{RT}\frac{\mathrm{d}p_K(t)}{\mathrm{d}t} + \frac{1}{|K|}\int_{\partial K} \boldsymbol{q}(\boldsymbol{x},t)\cdot\boldsymbol{n}_{\partial K}(\boldsymbol{x}) = f_K(t), \tag{2.42}$$

kde $\boldsymbol{n}_{\partial K}(\boldsymbol{x})$ označuje jednotkový vektor vnější normály v bodě $\boldsymbol{x} \in \partial K$ vzhledem k elementu K. Pro časovou diskretizaci semi-diskrétního schématu (2.42) použijeme zpětnou Eulerovu metodu. Tím získáme diskrétní schéma

$$\frac{\Phi_K M}{RT}\frac{p_K^{i+1} - p_K^i}{\Delta t_i} + \frac{1}{|K|}\sum_{E\in\mathcal{E}_K}\underbrace{\int_E \boldsymbol{q}^{i+1}\cdot\boldsymbol{n}_{K,E}}_{=q_{K,E}^{i+1}} = f_K^{i+1}, \qquad \forall K \in \mathcal{K}_h, \tag{2.43}$$

kde $\boldsymbol{n}_{K,E}$ označuje jednotkový vektor vnější normály na hraně E vzhledem k elementu K. Koeficienty $q_{K,E}^{i+1}$ vystupující v rovnici (2.43) odpovídají toku funkce $\boldsymbol{q}$ přes hranu E ve směru vnější normály vzhledem k elementu K v čase $t = t_{i+1}$.

2.4 Sestavení hlavní soustavy rovnic

Pro spojení rovnic (2.34) a (2.43) potřebujeme vyjádřit hodnoty koeficientů $q_{K,E}$ v čase $t = t_{i+1}$. Nelineární vztah (2.34) linearizujeme v čase pomocí metody zamrzlých koeficientů. Při výpočtu hodnot $q_{K,E}^{i+1}$ využijeme hodnoty $\widetilde{A}_{K,E,F}^i$ a $\widetilde{G}_{K,E}^i$ závisející na p_K^i z předchozí časové hladiny, čímž získáme lineární vztah pro p_K^{i+1} a p_E^{i+1}:

$$q_{K,E}^{i+1} = \sum_{F\in\mathcal{E}_K}\widetilde{A}_{K,E,F}^i\left(p_K^{i+1} - p_F^{i+1} + \widetilde{G}_{K,F}^i\right), \qquad \forall K \in \mathcal{K}_h,\ \forall E \in \mathcal{E}_K. \tag{2.44}$$

Proto diskrétní schéma (2.43) označujeme jako semi-implicitní.

Dále směřujeme k sestavení soustavy rovnic pro neznámé p_K^{i+1}, p_E^{i+1}. K tomu nám poslouží vztah (2.20) platný pro vnitřní hrany $E \in \mathcal{E}_h^{int}$ nezávisle na časové hladině. Potřebné vztahy pro okrajové hrany $E \in \mathcal{E}_h^{ext}$ získáme diskretizací okrajových podmínek.

[4] Cell-Centered Finite Volume Method

2.4.1 Diskretizace okrajových podmínek

Při diskretizaci okrajových podmínek předpokládáme, že na všech vnějších hranách $E \in \mathcal{E}_h^{ext}$ je dána podmínka právě jednoho typu, platí tedy buď $E \subset \Gamma_{Dir}$ nebo $E \subset \Gamma_{Neu}$. Tento předpoklad lze chápat jako diskretizaci podmínky (1.9). Dále označíme $\mathcal{E}_h^{Dir}$ resp. $\mathcal{E}_h^{Neu}$ množiny hran, na kterých je předepsaná podmínka Dirichletova resp. Neumannova typu.

Podle (2.19) mají koeficienty $q_{K,E}$ význam toku funkce $\boldsymbol{q}$ přes hranu E ve směru vnější normály. Neumannovu podmínku (1.8c) na hraně $E \in \mathcal{E}_h^{Neu}$ tedy můžeme vyjádřit pomocí integrálu

$$q_{K,E}^i = \int_E q_{Neu}(\boldsymbol{x}, t_i) \overset{\text{ozn}}{=} q_{Neu}^i(E). \tag{2.45}$$

V případě Dirichletovy podmínky (1.8b) na hraně $E \in \mathcal{E}_h^{Dir}$ můžeme v souladu s (2.24) přímo spočítat stopu tlaku na hraně E:

$$p_E^i = \frac{1}{|E|} \int_E p_{Dir}(\boldsymbol{x}, t_i) \overset{\text{ozn}}{=} p_{Dir}^i(E). \tag{2.46}$$

2.4.2 Redukce soustavy

Spojením diskretizované transportní rovnice (2.43), diskrétního Darcyho zákona (2.44), bilance toku na vnitřních hranách (2.20) a okrajových podmínek (2.45) a (2.46) získáme soustavu $N_K + N_E$ lineárních rovnic pro stejný počet neznámých p_K^{i+1}, p_E^{i+1}, kterou při označení

$$\lambda_K^i = \frac{\Phi_K M}{RT} \frac{|K|}{\Delta t_i}, \qquad F_K^i = |K| f_K^i, \tag{2.47}$$

zapíšeme ve tvaru

$$\lambda_K^i \left(p_K^{i+1} - p_K^i \right) + \sum_{E \in \mathcal{E}_K} \sum_{F \in \mathcal{E}_K} \widetilde{A}_{K,E,F}^i \left(\widetilde{G}_{K,F}^i - p_F^{i+1} + p_K^{i+1} \right) = F_K^{i+1}, \qquad \forall K \in \mathcal{K}_h, \tag{2.48a}$$

$$\sum_{K: E \in \mathcal{E}_K} \sum_{F \in \mathcal{E}_K} \widetilde{A}_{K,E,F}^i \left(\widetilde{G}_{K,F}^i - p_F^{i+1} + p_K^{i+1} \right) = 0, \qquad \forall E \in \mathcal{E}_h^{int}, \tag{2.48b}$$

$$\sum_{\substack{F \in \mathcal{E}_K, \\ K: E \in \mathcal{E}_K}} \widetilde{A}_{K,E,F}^i \left(\widetilde{G}_{K,F}^i - p_F^{i+1} + p_K^{i+1} \right) = q_{Neu}^{i+1}(E), \quad \forall E \in \mathcal{E}_h^{Neu}, \tag{2.48c}$$

$$p_E^{i+1} = p_{Dir}^{i+1}(E), \quad \forall E \in \mathcal{E}_h^{Dir}. \tag{2.48d}$$

Tato soustava je však pro řešení zbytečně velká – můžeme totiž využít její blokové struktury a pomocí metody hybridizace snížit počet rovnic v úloze. Z rovnice (2.48a) vyjádříme p_K^{i+1}:

$$p_K^{i+1} = \frac{1}{\lambda_K^i + \widetilde{A}_K^i} \left(F_K^{i+1} + \lambda_K^i p_K^i - \sum_{F \in \mathcal{E}_K} \widetilde{A}_{K,F}^i \left(\widetilde{G}_{K,F}^i - p_F^{i+1} \right) \right), \tag{2.49}$$

kde jsme použili označení

$$\widetilde{A}_{K,E}^i = \sum_{F \in \mathcal{E}_K} \widetilde{A}_{K,E,F}^i, \qquad \widetilde{A}_K^i = \sum_{E \in \mathcal{E}_K} \widetilde{A}_{K,E}^i. \tag{2.50}$$

Dosazením do (2.44) dostaneme

$$\begin{aligned} q_{K,E}^{i+1} &= \sum_{F\in\mathcal{E}_K} \widetilde{A}_{K,E,F}^i \left(\widetilde{G}_{K,F}^i - p_F^{i+1}\right) + \widetilde{A}_{K,E}^i p_K^{i+1} \\ &= \sum_{F\in\mathcal{E}_K} \widetilde{A}_{K,E,F}^i \left(\widetilde{G}_{K,F}^i - p_F^{i+1}\right) + \frac{\widetilde{A}_{K,E}^i}{\lambda_K^i + \widetilde{A}_K^i} \left(F_K^{i+1} + \lambda_K^i p_K^i - \sum_{F\in\mathcal{E}_K} \widetilde{A}_{K,F}^i \left(\widetilde{G}_{K,F}^i - p_F^{i+1}\right)\right) \\ &= \sum_{F\in\mathcal{E}_K} \underbrace{\left(\widetilde{A}_{K,E,F}^i - \frac{\widetilde{A}_{K,E}^i \widetilde{A}_{K,F}^i}{\lambda_K^i + \widetilde{A}_K^i}\right)}_{\widetilde{B}_{K,E,F}^i} \left(\widetilde{G}_{K,F}^i - p_F^{i+1}\right) + \frac{\widetilde{A}_{K,E}^i}{\lambda_K^i + \widetilde{A}_K^i} \left(F_K^{i+1} + \lambda_K^i p_K^i\right). \end{aligned} \tag{2.51}$$

Pomocí (2.51) nyní stejným postupem, jakým byly odvozeny rovnice (2.48b) a (2.48c), získáme soustavu

$$\sum_{K:E\in\mathcal{E}_K} \left[\sum_{F\in\mathcal{E}_K} \widetilde{B}_{K,E,F}^i \left(\widetilde{G}_{K,F}^i - p_F^{i+1}\right) + \frac{\widetilde{A}_{K,E}^i}{\lambda_K^i + \widetilde{A}_K^i} \left(F_K^{i+1} + \lambda_K^i p_K^i\right)\right] = 0, \qquad \forall E \in \mathcal{E}_h^{int}, \tag{2.52a}$$

$$\sum_{\substack{F\in\mathcal{E}_K \\ K:E\in\mathcal{E}_K}} \widetilde{B}_{K,E,F}^i \left(\widetilde{G}_{K,F}^i - p_F^{i+1}\right) + \frac{\widetilde{A}_{K,E}^i}{\lambda_K^i + \widetilde{A}_K^i} \left(F_K^{i+1} + \lambda_K^i p_K^i\right) = q_{Neu}^{i+1}(E), \quad \forall E \in \mathcal{E}_h^{Neu}, \tag{2.52b}$$

$$p_E^{i+1} = p_{Dir}^{i+1}(E), \quad \forall E \in \mathcal{E}_h^{Dir}. \tag{2.52c}$$

Soustavu (2.48) o dimenzi $N_E + N_K$ jsme tedy metodou hybridizace převedli na soustavu (2.52) pro N_E neznámých p_E^{i+1} doplněnou o explicitní vztah (2.49) pro N_K neznámých p_K^{i+1}. Soustava (2.52) je řídká, řádky odpovídající rovnicím (2.52a) obsahují 7 nenulových prvků, řádky odpovídající rovnicím (2.52b) obsahují 4 nenulové prvky a řádky odpovídající rovnicím (2.52c) obsahují 1 nenulový prvek.

2.5 Algoritmus výpočtu

Diskrétní schéma (2.43) udává, jak spočítat aproximaci řešení p_h^{i+1} v čase t_{i+1} pomocí aproximace p_h^i z předchozí časové hladiny t_i, přičemž známe počáteční stav v čase t_0. Abychom získali aproximaci řešení v čase $t = T_f$, musíme iteračně určit aproximaci ve všech předchozích časových hladinách $t_i \in \mathcal{T}$. Podrobný algoritmus výpočtu numerického řešení pak vypadá takto:

Algoritmus 1

1. Zvolíme $h > 0$ a oblast $\Omega \subset \mathbb{R}^2$ pokryjeme strukturovanou konformní obdélníkovou sítí $\mathcal{K}_h$. Časový interval diskretizujeme pomocí množiny $\mathcal{T}$.
2. Načteme vstupní data charakterizující úlohu, např. porozita, permeabilita, počáteční podmínka a další parametry.
3. Diskretizací počáteční podmínky (1.8a) určíme střední hodnoty tlaku p_K^0 na jednotlivých elementech $K \in \mathcal{K}_h$ na výchozí časové hladině t_0. Podobně diskretizujeme okrajové podmínky pomocí (2.45) a (2.46).

4. Na každém elementu $K \in \mathcal{K}_h$ vypočteme dle (2.39) a (2.35) prvky lokální matice $\mathbf{A}_K$ a složky vektoru $\boldsymbol{G}_K$. Jsou konstantní v čase, proto je stačí vypočítat jen jednou.
5. Položíme $i = 0$.
6. Dokud $t_i < T_f$, opakujeme následující kroky:
   6.1. Sestavíme matici redukované soustavy (2.52) a vektor pravé strany.
   6.2. Řešíme soustavu N_E lineárních rovnic pro N_E neznámých p_E^{i+1}.
   6.3. Na každém elementu určíme hodnotu p_K^{i+1} podle (2.49).
   6.4. Přejdeme na následující časovou hladinu, $i \to i + 1$.
7. V čase $t_i = T_f$ získáme numerické řešení dané úlohy.

2.6 Upwindová stabilizace

Jak uvidíme v kapitole 4.3, numerické schéma popsané v předchozí části kapitoly 2 není stabilní pro silně advekční úlohy. Popsané numerické schéma proto modifikujeme pomocí metody *upwindu*, která je jednou z často používaných metod pro stabilizaci numerických schémat při řešení hyperbolických zákonů zachování.

V kapitole 2.3 jsme tok $\boldsymbol{q} = \rho\boldsymbol{u}$ aproximovali přímo pomocí funkce $\boldsymbol{q}_h$ z Raviartova–Thomasova prostoru, nyní však prostorovou diskretizaci provedeme v závislosti na směru toku přes elementy sítě. Darcyho rychlost $\boldsymbol{u}$ aproximujeme pomocí funkce $\boldsymbol{u}_h \in \mathbf{RT}_0(\mathcal{K}_h)$, stejným postupem jako v kapitole 2.2 tak při vyjádření

$$\boldsymbol{u}_h|_K = \sum_{E \in \mathcal{E}_K} u_{K,E} \boldsymbol{\omega}_{K,E}, \tag{2.53}$$

kde $\boldsymbol{\omega}_{K,E}$ jsou bazické funkce prostoru $\mathbf{RT}_0(K)$, dostaneme diskrétní Darcyho zákon ve tvaru

$$u_{K,E} = \sum_{F \in \mathcal{E}_K} A_{K,E,F} \left(p_K - p_F + \widetilde{G}_{K,F} \right), \qquad \forall K \in \mathcal{K}_h,\ \forall E \in \mathcal{E}_K, \tag{2.54}$$

a bilanční podmínku pro numerický tok funkce $\boldsymbol{u}_h$ na vnitřních hranách sítě $\mathcal{K}_h$

$$u_{K_1,E} + u_{K_2,E} = 0, \qquad \forall E \in \mathcal{E}_h, E \in \mathcal{E}_{K_1} \cap \mathcal{E}_{K_2}. \tag{2.55}$$

Náhradou $\boldsymbol{q} = \rho\boldsymbol{u}$ v rovnici (2.41) dostaneme

$$\frac{M}{RT} \frac{\mathrm{d}}{\mathrm{d}t} \int_K \Phi(\boldsymbol{x}) p(\boldsymbol{x},t) + \int_K \nabla \cdot (\rho(\boldsymbol{x},t)\boldsymbol{u}(\boldsymbol{x},t)) = \int_K f(\boldsymbol{x},t). \tag{2.56}$$

Dále postupujeme stejně jako v kapitole 2.3. Náhradou prvního integrálu na levé straně a integrálu na pravé straně rovnice (2.56) středními hodnotami příslušných veličin a úpravou druhého integrálu na levé straně rovnice (2.56) pomocí Greenovy formule dostaneme

$$\frac{\Phi_K M}{RT} \frac{\mathrm{d}p_K(t)}{\mathrm{d}t} + \frac{1}{|K|} \sum_{E \in \mathcal{E}_K} \rho_E^{upw}(t) \underbrace{\int_E \boldsymbol{u}(\boldsymbol{x},t) \cdot \boldsymbol{n}_{K,E}(\boldsymbol{x})}_{u_{K,E}(t)} = f_K(t), \tag{2.57}$$

kde $\rho_E^{upw}(t)$ představuje upwindovou hodnotu hustoty na hraně E, jejíž volbu upřesníme později. Nelineární semi-diskrétní schéma (2.57) diskretizujeme v čase pomocí zpětné Eulerovy metody a linearizujeme pomocí metody zamrzlých koeficientů. Tím získáme diskrétní schéma

$$\frac{\Phi_K M}{RT}\frac{p_K^{i+1}-p_K^i}{\Delta t_i}+\frac{1}{|K|}\sum_{E\in\mathcal{E}_K}\rho_E^{i,upw}u_{K,E}^{i+1}=f_K^{i+1},\qquad \forall K\in\mathcal{K}_h. \tag{2.58}$$

Nyní již můžeme upřesnit volbu koeficientů $\rho_E^{i,upw}$, které volíme dle směru numerického toku funkce $\boldsymbol{u}_h$ přes hranu E na stejné časové hladině:

$$\rho_E^{i,upw}=\begin{cases}\rho_{K_1}^i & \text{pokud } u_{K_1,E}^i\geq 0,\\ \rho_{K_2}^i & \text{jinak,}\end{cases}\qquad \forall E\in\mathcal{E}_h^{int}, E\in\mathcal{E}_{K_1}\cap\mathcal{E}_{K_2}, \tag{2.59a}$$

$$\rho_E^{i,upw}=\begin{cases}\rho_{K_1}^i & \text{pokud } u_{K_1,E}^i\geq 0,\\ \rho_{Dir}^i(E) & \text{jinak,}\end{cases}\qquad \forall E\in\mathcal{E}_h^{ext}, E\in\mathcal{E}_{K_1}, \tag{2.59b}$$

kde $\rho_{K_\iota}^i\equiv\rho(p_{K_\iota}^i)$ a $\rho_{Dir}^i(E)\equiv\rho(p_{Dir}^i(E))$ představuje střední hodnotu hustoty na hraně E předepsanou dle Dirichletovy okrajové podmínky (2.46). Zde je nutné podotknout, že pro korektnost volby upwindové hodnoty na hranici oblasti Γ je potřeba znát hodnotu hustoty kromě Dirichletovy části hranice Γ_{Dir} také na části Neumannovy hranice Γ_{Neu}^*, kde tekutina vtéká dovnitř oblasti, tedy

$$\Gamma_{Neu}^*=\{\boldsymbol{x}\in\Gamma_{Neu}\mid \boldsymbol{q}(\boldsymbol{x},t)\cdot\boldsymbol{n}(\boldsymbol{x})<0\}. \tag{2.60}$$

Spojením diskretizované rovnice kontinuity (2.58), bilanční podmínky (2.55) a okrajových podmínek (2.45) a (2.46) dostaneme soustavu N_K+N_E lineárních rovnic pro stejný počet neznámých p_K^{i+1}, p_E^{i+1}, kterou při značení (2.47) zapíšeme ve tvaru

$$\lambda_K^i\left(p_K^{i+1}-p_K^i\right)+\sum_{E\in\mathcal{E}_K}\rho_E^{i,upw}\sum_{F\in\mathcal{E}_K}A_{K,E,F}^i\left(\widetilde{G}_{K,F}^i-p_F^{i+1}+p_K^{i+1}\right)=F_K^{i+1},\qquad \forall K\in\mathcal{K}_h, \tag{2.61a}$$

$$\sum_{K:E\in\mathcal{E}_K}\sum_{F\in\mathcal{E}_K}A_{K,E,F}\left(\widetilde{G}_{K,F}^i-p_F^{i+1}+p_K^{i+1}\right)=0,\qquad \forall E\in\mathcal{E}_h^{int}, \tag{2.61b}$$

$$\sum_{\substack{F\in\mathcal{E}_K,\\ K:E\in\mathcal{E}_K}}\rho_K^i A_{K,E,F}\left(\widetilde{G}_{K,F}^i-p_F^{i+1}+p_K^{i+1}\right)=q_{Neu}^{i+1}(E),\quad \forall E\in\mathcal{E}_h^{Neu}, \tag{2.61c}$$

$$p_E^{i+1}=p_{Dir}^{i+1}(E),\quad \forall E\in\mathcal{E}_h^{Dir}. \tag{2.61d}$$

Podobně jako v kapitole 2.4.2 pomocí metody hybridizace redukujeme soustavu (2.61). Z rovnice (2.61a) vyjádříme p_K^{i+1}:

$$p_K^{i+1}=\frac{1}{\lambda_K^i+\widetilde{A}_K^i}\left(F_K^{i+1}+\lambda_K^i p_K^i-\sum_{F\in\mathcal{E}_K}\rho_F^{i,upw}A_{K,F}\left(\widetilde{G}_{K,F}^i-p_F^{i+1}\right)\right), \tag{2.62}$$

kde jsme využili aproximace $\mathbf{A}_K \approx \mathbf{A}_K^{(\ell)}$, kde $\mathbf{A}_K^{(\ell)} = \mathrm{diag}\{A_{K,F}\}_{F\in\mathcal{E}_K}$ je dle (2.39) diagonální matice. Dále jsme použili označení

$$\widetilde{A}_K^i = \sum_{E\in\mathcal{E}_K} \rho_E^{i,upw} A_{K,E}. \tag{2.63}$$

Dosazením (2.62) do rovnic (2.61b) a (2.61c) dostaneme podobně jako v kapitole 2.4.2 finální soustavu lineárních rovnic pro p_E^{i+1}:

$$\sum_{K:E\in\mathcal{E}_K}\left[\sum_{F\in\mathcal{E}_K} B_{K,E,F}^i\left(\widetilde{G}_{K,F}^i - p_F^{i+1}\right) + \frac{A_{K,E}}{\lambda_K^i + \widetilde{A}_K^i}\left(F_K^{i+1} + \lambda_K^i p_K^i\right)\right] = 0, \qquad \forall E\in\mathcal{E}_h^{int}, \tag{2.64a}$$

$$\rho_K^i\left[\sum_{\substack{F\in\mathcal{E}_K\\ K:E\in\mathcal{E}_K}} B_{K,E,F}^i\left(\widetilde{G}_{K,F}^i - p_F^{i+1}\right) + \frac{A_{K,E}}{\lambda_K^i + \widetilde{A}_K^i}\left(F_K^{i+1} + \lambda_K^i p_K^i\right)\right] = q_{Neu}^{i+1}(E), \quad \forall E\in\mathcal{E}_h^{Neu}, \tag{2.64b}$$

$$p_E^{i+1} = p_{Dir}^{i+1}(E), \quad \forall E\in\mathcal{E}_h^{Dir}, \tag{2.64c}$$

kde

$$B_{K,E,F}^i = A_{K,E,F} - \frac{A_{K,E}A_{K,F}}{\lambda_K^i + \widetilde{A}_K^i}\rho_F^{i,upw}. \tag{2.65}$$

2.6.1 Algoritmus

Numerické schéma s upwindovou stabilizací je modifikací základní varianty numerického schématu, kterému jsme se věnovali v převážné části kapitoly 2. Algoritmus výpočtu numerického řešení při použití upwindové stabilizace se proto oproti algoritmu 1 změní jen v některých bodech, které jsou popsány níže:

- Při inicializaci ve 3. bodu algoritmu je potřeba inicializovat upwindové hodnoty hustoty $\rho_E^{0,upw}$ na hranách sítě. Počáteční hodnoty zvolíme takto:

$$\rho_E^{0,upw} = \begin{cases} \frac{1}{2}\left(\rho_{K_1}^0 + \rho_{K_2}^0\right) & \text{pro } E\in\mathcal{E}_h^{int}, E\in\mathcal{E}_{K_1}\cap\mathcal{E}_{K_2},\\ \rho_{K_1}^0 & \text{pro } E\in\mathcal{E}_h^{ext}, E\in\mathcal{E}_{K_1}, \end{cases} \tag{2.66}$$

  kde $\rho_{K_\iota}^0 \equiv \rho\left(p_{K_\iota}^0\right)$, $\iota = 1,2$ označuje hustotu na elementu K_ι získanou pomocí počáteční podmínky $p_{K_\iota}^0$.

- Při sestavování matice soustavy v bodu 6.1. místo koeficientů $\widetilde{B}_{K,E,F}^i$ používáme koeficienty $B_{K,E,F}^i$ a příslušné vztahy (2.63) a (2.65).

- Při aktualizaci středních hodnot tlaku na elementech v bodu 6.3. používáme místo vztahu (2.49) modifikovaný vztah (2.62).

- Před přechodem na následující časovou hladinu t_{i+1} vypočteme dle definičních vztahů (2.59) upwindové hodnoty $\rho_E^{i+1,upw}$ pro novou časovou hladinu.

Kapitola 3

Implementace

Tato kapitola poskytuje podrobný popis implementace numerického schématu pro řešení úlohy (1.7) popsaného v kapitole 2. V podkapitole 3.1 se nejprve zabýváme základním řešičem využívajícím knihovnu UMFPACK [6, 5, 7] pro řešení řídkých soustav lineárních rovnic pomocí přímých metod. V následující podkapitole 3.2 popíšeme implementaci řešiče pomocí knihovny TNL, která poskytuje efektivní iterační řešiče pro řešení řídkých soustav lineárních rovnic. TNL současně poskytuje jednotné rozhraní pro paralelní architekturu CUDA a umožňuje tak psát programy spustitelné na CPU i na GPU.

S ohledem na rozdíly mezi variantami numerického schématu popsané v kapitole 2.6.1 se při popisu implementace omezíme na základní schéma bez upwindové stabilizace, modifikované schéma je implementováno velmi podobně. Primárním cílem této kapitoly je poskytnout uživatelský manuál ke kódu dostupnému k této práci v digitální podobě. Komentáře v kódu a názvy funkcí jsou v angličtině a měly by být intuitivní, tudíž nejsou blíže v této práci vysvětlovány. Při popisu je kladen důraz na podstatné rozdíly mezi implementacemi využívajících různých knihoven.

3.1 Řešič využívající knihovnu UMFPACK

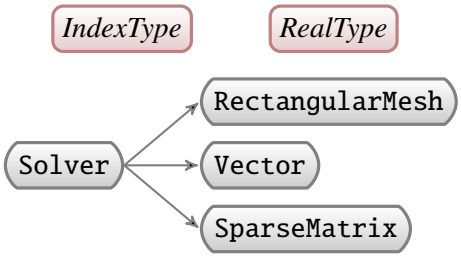
Nejprve popíšeme základní řešič implementovaný pomocí jazyka C++ a knihovny UMFPACK, který primárně sloužil jako odrazový můstek pro vývoj netriviálního řešiče s využitím knihovny TNL. Postupem času však z důvodu testování bylo potřeba v určitých věcech napodobit TNL, aby bylo možné srovnání obou řešičů. Základní struktura řešiče při použití knihovny TNL je však velmi odlišná a popis základního řešiče tak umožní zdůraznit hlavní rozdíly mezi jednotlivými implementacemi.

Pro obecnost kódu v hlavním hlavičkovém souboru definujeme vlastní typy IndexType a RealType, které bude možné měnit podle potřeby:

```
1 typedef double RealType;
2 typedef int IndexType;
```

Základem řešiče je třída Solver, která je implementací algoritmu popsaného v kapitole 2.5. Pro práci s daty dále definujeme pomocné třídy. Vlastnosti strukturované ekvidistantní obdélníkové sítě vyjádříme pomocí třídy RectangularMesh, hodnoty diskrétních veličin uložíme v objektech typu Vector a hlavní matici soustavy sestavíme v objektu typu SparseMatrix. Vazby mezi jednotlivými objekty jsou schematicky znázorněny pomocí diagramu na obr. 3.1.

Řešič spouštíme pomocí metody Solver::run, kde se provede inicializace a spuštění výpočtu. Další důležité metody objektu Solver představíme pomocí algoritmu 2, který je modifikací algoritmu 1 s ohledem na datové struktury. Oproti skutečné implementaci v algoritmu 2 pomineme některé technické detaily jako např. ukládání snapshotů v průběhu výpočtu.



Obrázek 3.1: Vazby mezi hlavními objekty a typy v řešiči.

Algoritmus 2

1. Pro zvolené parametry $N_x, N_y \in \mathbb{N}$ vytvoříme objekt `RectangularMesh` (viz 3.1.1) reprezentující numerickou síť $\mathcal{K}_h$ o rozměrech $N_x \times N_y$ na obdélníkové oblasti Ω. Symboly h_x, h_y označíme rozměry elementů v příslušném směru a volíme parametr sítě $h = \min\{h_x, h_y\}$.

2. Vstupní data charakterizující úlohu načteme jako atributy objektu `Solver`, např. `Solver::porosity`, `Solver::permeability`, `Solver::idealGasCoefficient` $:= \frac{M}{RT}$.

3. Diskretizovanou počáteční podmínku (1.8a) načteme do vektoru `Solver::pressure`, který v průběhu výpočtu reprezentuje střední hodnoty tlaku p_K^i v čase t_i na jednotlivých elementech. Okrajové podmínky diskretizované pomocí (2.46) resp. (2.45) reprezentujeme pomocí síťových funkcí `get_dirichlet_value` resp. `get_neumann_value`.

4. Prvky lokálních matic $\mathbf{A}_K$ na jednotlivých elementech $K \in \mathcal{K}_h$ vypočteme dle (2.39) v rámci metody `Solver::update_auxiliary_vectors` (viz 3.1.3). Složky vektorů $\boldsymbol{G}_K$ počítáme dle (2.35) pomocí metody `Solver::G_KE`.

5. Položíme $i = 0$.

6. Dokud $t_i < T_f$, opakujeme následující kroky:

   6.1. Pomocí metody `Solver::update_auxiliary_vectors` (viz 3.1.3) aktualizujeme prvky pomocných vektorů.

   6.2. V metodě `Solver::update_main_system` (viz 3.1.4) sestavíme matici soustavy (2.52) reprezentovanou objektem `Solver::mainMatrix` a vektor pravé strany `Solver::rhs`.

   6.3. Soustavu lineárních rovnic pro neznámé p_E^{i+1} z předchozího bodu řešíme pomocí metody `mainMatrix.linear_solve` (viz 3.1.2), řešení uložíme do vektoru `Solver::ptrace`.

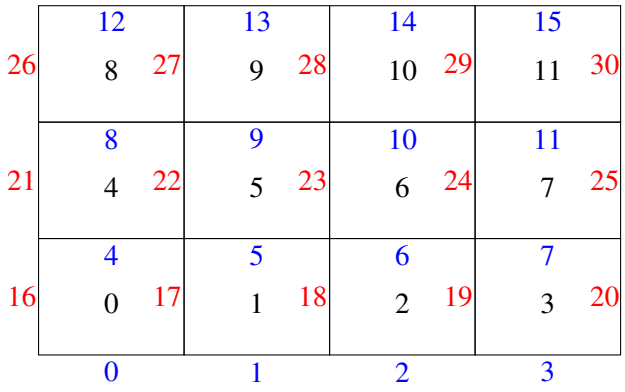
   6.4. Pomocí metody `Solver::update_pressure` (viz 3.1.5) s využitím (2.49) aktualizujeme střední hodnoty tlaku na elementech.

   6.5. Přejdeme na následující časovou hladinu, $i \to i + 1$.

7. V čase $t_i = T_f$ získáme numerické řešení dané úlohy.

3.1.1 Reprezentace numerické sítě

Elementy sítě očíslujeme postupně indexy $K \in \{0, \ldots, N_K - 1\}$ a hrany sítě očíslujeme analogicky indexy $E \in \{0, \ldots, N_E - 1\}$. Při číslování elementů a hran ekvidistantní obdélníkové sítě postupujeme



Obrázek 3.2: Očíslování elementů a hran sítě o rozměrech 4×3 elementy. Elementy jsou očíslovány černě, horizontální hrany modře a vertikální hrany červeně.

po řádcích, přičemž v případě hran nejprve očíslujeme horizontální hrany a poté vertikální hrany.[1] Konkrétní očíslování sítě 4×3 elementy je znázorněno na obr. 3.2.

Hodnoty síťových funkcí uložíme pomocí vektorů nebo matic indexovaných pomocí indexů K a E. Síťové funkce tedy nejsou součástí objektu `RectangularMesh`, který pouze popisuje grafové vlastnosti sítě. Konkrétně budeme potřebovat určit tyto vlastnosti:

- zda je daná hrana E horizontální či vertikální,
- zda je daná hrana E vnitřní nebo okrajová,
- pro danou hranu E určit k ní přilehlé elementy K_1 a K_2,
- pro daný element K určit sousední hrany E_1, E_2, E_3, E_4.

Uvažujme síť o rozměrech $N_x \times N_y$ elementů obsahující $N_x N_y$ elementů, $N_x(N_y + 1)$ horizontálních hran a $(N_x + 1)N_y$ vertikálních hran. Vzhledem k výše uvedenému číslování je hrana E horizontální, právě když $E < N_x(N_y + 1)$.

Dále můžeme každý element K, každou horizontální hranu E_h a každou vertikální hranu E_v jednoznačně charakterizovat souřadnicemi (i, j), kde meze pro $i, j \in \mathbb{N}_0$ jsou

$$i < N_x, \quad j < N_y, \quad \text{pro element } K, \tag{3.1a}$$
$$i < N_x, \quad j < N_y + 1, \quad \text{pro hranu } E_h, \tag{3.1b}$$
$$i < N_x + 1, \quad j < N_y, \quad \text{pro hranu } E_v. \tag{3.1c}$$

Element $K = 0$ a horizontální hrana $E_h = 0$ mají souřadnice $(0, 0)$, vertikální hrana $E_v = N_x(N_y + 1)$ má také souřadnice $(0, 0)$.

Při určování zbývajících tří vlastností využijeme charakterizace hran a elementů pomocí souřadnic. Horizontální hrana je okrajová, právě když její souřadnice $j = 0$ nebo $j = N_y$. Podobně vertikální hrana je okrajová, právě když její souřadnice $i = 0$ nebo $i = N_x$. Pro určení sousedních elementů k dané hraně nejprve pomocí globálního indexu E určíme souřadnice hrany a poté souřadnice sousedního elementu, pomocí kterých určíme index K. Souřadnice (i, j) vnitřní horizontální resp. vertikální hrany E jsou současně souřadnicemi sousedního elementu směrem nahoru resp. doprava a souřadnice druhého sousedního elementu jsou $(i, j - 1)$ resp. $(i - 1, j)$. V případě okrajových hran stačí kontrolovat meze

[1] V knihovně `TNL` se postupuje opačně, nejprve jsou očíslovány vertikální a poté horizontální hrany.

pro souřadnice i, j. Podobně postupujeme při určování sousedních hran daného elementu. Modifikací souřadnic elementu získáme souřadnice hran, ze kterých dopočítáme globální index.

3.1.2 Uložení řídké matice

Pro efektivní uložení řídké matice stačí ukládat pouze nenulové prvky. Pro objekt `SparseMatrix` použijeme formát CSR[2], který ukládá informace o nenulových prvcích matice $\mathbf{A} = [a_{i,j}]_{i=0}^{m-1}{}_{j=0}^{n-1}$ pomocí tří vektorů:

- Vektor `values` obsahuje hodnoty $a_{i,j}$ nenulových prvků matice, uspořádaných po řádcích.
- Vektor `column_indexes` obsahuje sloupcové indexy každého nenulového prvku. Tedy pokud $\texttt{values}(k) = a_{i,j}$, pak $\texttt{column\_indexes}(k) = j$.
- Vektor `row_pointers` obsahuje na i-té pozici umístění prvního nenulového prvku na i-tém řádku ve vektoru `values`. Tedy pokud $\texttt{values}(k) = a_{i,j}$, pak

$$\texttt{row\_pointers}(i) \leq k < \texttt{row\_pointers}(i+1). \tag{3.2}$$

  Dle konvence dále klademe $\texttt{row\_pointers}(m) = N_{NZ}$, kde N_{NZ} je počet nenulových prvků matice $\mathbf{A}$.

Pro uložení matice $\mathbf{A}$ tedy stačí uložit N_{NZ} prvků typu `RealType` a $N_{NZ} + m + 1$ prvků typu `IndexType`.

Pro přečtení prvku $a_{i,j}$ z formátu CSR nejprve pomocí (3.2) pro daný index i určíme úsek vektoru `column_indexes`, ve kterém hledáme daný index j. Pokud index j nenajdeme, je $a_{i,j} = 0$, v opačném případě nenulovou hodnotu $a_{i,j}$ najdeme na odpovídající pozici vektoru `values`.

Sestavování matice ve formátu CSR spočívá v přidávání prvků do vektorů `values` a `column_indexes` a v aktualizaci vektoru `row_pointers` po každém přidání nového prvku. Z hlediska efektivity je tedy vhodné využít předem známých počtů nenulových prvků na jednotlivých řádcích a vektor `row_pointers` nastavit předem. V neposlední řadě dopředu alokujeme vektory `values` a `column_indexes` na požadovanou velikost N_{NZ}.

V objektu `SparseMatrix` definujeme metodu `linear_solve`, která řeší soustavu lineárních rovnic $\mathbf{A}\boldsymbol{x} = \boldsymbol{b}$ s danou maticí $\mathbf{A}$ a pravou stranou $\boldsymbol{b}$. Při tom využijeme knihovnu `UMFPACK`, která však interně pracuje s maticí ve formátu CSC[3], ve kterém jsou oproti formátu CSR nenulové prvky uspořádány po sloupcích. Reprezentace matice $\mathbf{A}$ uložené ve formátu CSC a matice $\mathbf{A}^T$ uložené ve formátu CSR jsou ekvivalentní, transpozici však neprovádíme ručně a pomocí přepínače nastavíme `UMFPACK` tak, aby se řešila soustava s transponovanou maticí. Místo soustavy $\mathbf{A}\boldsymbol{x} = \boldsymbol{b}$ s maticí $\mathbf{A}$ ve formátu CSC tedy řešíme soustavu $\mathbf{A}^T\boldsymbol{x} = \boldsymbol{b}$ s maticí ve formátu CSR.

3.1.3 Výpočet pomocných vektorů

Při popisu algoritmu 2 nejprve definujeme pomocnou metodu `update_auxiliary_vectors` objektu `Solver` pro výpočet pomocných vektorů:

```
1 bool Solver::update_auxiliary_vectors( const RealType & time, const RealType & tau )
2 {
```

V této metodě budeme v každém časovém kroku počítat koeficienty λ_K^i podle (2.47), které uložíme do vektoru `Solver::lambda`. Elementy sítě indexujeme pomocí proměnné `cell`.

[2] Compressed Sparse Row
[3] Compressed Sparse Column

```
    for( IndexType cell = 0; cell < mesh.num_cells(); cell++ ) {
        lambda[ cell ] = porosity[ cell ] * idealGasCoefficient * mesh.cell_volume( cell ) /
        ↪  tau;
    }
```

Pro výpočet koeficientů $\widetilde{A}^i_{K,E}$ a $\widetilde{A}^i_K$ definujeme koeficienty $\beta_{K,E}$ a α_K tak, že

$$\widetilde{A}^i_K = p^i_K \alpha_K, \tag{3.3a}$$

$$\widetilde{A}^i_{K,E} = p^i_K \beta_{K,E}. \tag{3.3b}$$

Koeficienty α_K a $\beta_{K,E}$ jsou konstantní v čase, proto je stačí spočítat pouze jednou. Hodnoty α_K uložíme do vektoru `Solver::alpha` indexovaného pomocí proměnné `cell` a hodnoty $\alpha_{K,E}$ uložíme ve formátu CSR do matice `Solver::beta` indexované pomocí proměnných `cell` a `edge`. K-tý řádek matice `beta` obsahuje právě 4 nenulové prvky $\beta_{K,E}, E \in \mathcal{E}_K$.

```
    if( time > initial_time )
        return true;

    for( IndexType cell = 0; cell < mesh.num_cells(); cell++ ) {
        alpha[ cell ] = 0.0;
        for( int i = 0; i < 4; i++ ) {
            IndexType edge = mesh.edge_for_cell( cell, i );
            RealType value = 2 * idealGasCoefficient * permeability[ cell ] / viscosity;
            if( mesh.is_horizontal_edge(edge) )
                value *= mesh.get_hx() / mesh.get_hy();
            else
                value *= mesh.get_hy() / mesh.get_hx();
            beta.setElement( cell, edge, value );
            alpha[ cell ] += value;
        }
    }

    return true;
}
```

3.1.4 Sestavení matice soustavy

V bodu 6.2. algoritmu 2 voláme metodu `Solver::update_main_system` pro sestavení matice soustavy `Solver::mainMatrix` a vektoru pravé strany `Solver::rhs`. Matici sestavujeme po řádcích odpovídajících jednotlivým hranám sítě, tohoto postup dále využijeme v kapitole 3.2.1 pro paralelizaci numerického schématu.

Předpokládáme, že `mainMatrix` je na začátku sestavování nulová matice, vektor `rhs` může obsahovat hodnoty z předchozího časového kroku. Pro každou rovnici soustavy (2.52) tedy nejprve vynulujeme příslušný prvek vektoru pravé strany.

```
bool Solver::update_main_system( const RealType & time )
{
    for( IndexType indexRow = 0; indexRow < mesh.num_edges(); indexRow++ ) {
        rhs[ indexRow ] = 0.0;
```

Počet nenulových prvků na daném řádku matice závisí na tom, zda je hrana odpovídající danému řádku vnitřní nebo je na ní předepsaná Dirichletova nebo Neumannova podmínka. Podle (2.52a) a (2.52b)

přispívají rovnice odpovídající Neumannovým okrajovým podmínkám do matice soustavy stejnými členy jako rovnice odpovídající vnitřním hranám, kde se ale sčítá přes oba sousední elementy. Tyto příspěvky nastavíme pomocí společného kódu, přičemž využijeme toho, že metoda `mesh.cell_for_edge()` vrací zápornou hodnotu, pokud přistupujeme k okrajové hraně z vnějšku.

```
        if( ! mesh.is_dirichlet_boundary( indexRow ) ) {
            for( IndexType i = 0; i < 2; i++ ) {
                IndexType cell = mesh.cell_for_edge( indexRow, i );
                // skip undefined element on Neumann boundary
                if( cell < 0 )
                    continue;
```

Uvnitř cyklu sčítajícího přes elementy dále sčítáme přes sousední hrany, jejichž indexy odpovídají sloupcovým indexům nenulových prvků na daném řádku. Dále přičteme příspěvky k vektoru pravé strany.

```
                for( IndexType j = 0; j < 4; j++ ) {
                    IndexType indexColumn = mesh.edge_for_cell( cell, j );

                    RealType B_KEF = -beta.getElement( cell, indexRow ) * pressure[ cell ] *
                                    beta.getElement( cell, indexColumn ) * pressure[ cell ] /
                                    ( lambda[ cell ] + alpha[ cell ] * pressure[ cell ] );
                    if( indexRow == indexColumn )
                        B_KEF += beta.getElement( cell, indexRow ) * pressure[ cell ];

                    // set main matrix element
                    RealType value = mainMatrix.getElement( indexRow, indexColumn );
                    mainMatrix.setElement( indexRow, indexColumn, value + B_KEF );

                    // right hand side
                    rhs[ indexRow ] += B_KEF * G_KE( cell, indexColumn ) * pressure[ cell ];
                }

                // right-hand-side
                rhs[ indexRow ] += beta.getElement( cell, indexRow ) * pressure[ cell ] /
                                ( lambda[ cell ] + alpha[ cell ] * pressure[ cell ] ) *
                                ( F[ cell ] + lambda[ cell ] * pressure[ cell ] );
            }
        }
```

Na závěr nastavíme příspěvky Dirichletových a Neumannových okrajových podmínek pomocí síťových funkcí `get_dirichlet_value` a `get_neumann_value`, které pro daný index hrany a danou časovou hladinu vrací hodnoty funkcí p_{Dir} resp. q_{Neu} diskretizovaných podle (2.46) resp. (2.45).

```
        // Dirichlet boundary
        else {
            mainMatrix.setElement( indexRow, indexRow, 1.0 );
            rhs[ indexRow ] = get_dirichlet_value( indexRow, time );
        }
        // Neumann boundary
        if( mesh.is_neumann_boundary( indexRow ) ) {
            rhs[ indexRow ] += get_neumann_value( indexRow , time );
        }
    }
    return true;
}
```

3.1.5 Výpočet tlaku na elementech

Vektor `Solver::pressure` obsahuje v i-tém kroku algoritmu 2 střední hodnoty tlaku p_K^i na jednotlivých elementech. Při přechodu z časové hladiny t_i na hladinu t_{i+1} spočítáme v bodu 6.4. algoritmu 2 hodnoty p_K^{i+1} podle rovnice (2.49), která představuje explicitní závislost střední hodnoty tlaku p_K^{i+1} na hodnotě p_K^i z předchozí časové hladiny. Navíc střední hodnota tlaku na elementu K na nové časové hladině t_{i+1} nezávisí na hodnotách na sousedních elementech K' z předchozí časové hladiny t_i, tedy hodnoty p_K^{i+1} stačí uložit přímo do vektoru `Solver::pressure`. Výpočet provedeme pro každý element K v metodě `Solver::update_pressure`.

```
1  bool Solver::update_pressure( void )
2  {
3      for( IndexType cell = 0; cell < mesh.num_cells(); cell++ ) {
4          RealType p = 0.0;
5          for( IndexType i = 0; i < mesh.edges_per_cell(); i++ ) {
6              IndexType edge = mesh.edge_for_cell( cell, i );
7              p += beta.getElement( cell, edge ) * ( ptrace[ edge ] - G_KE( cell, edge ) *
                   ↪  pressure[ cell ] );
8          }
9          p *= pressure[ cell ];
10         p += F[ cell ] + lambda[ cell ] * pressure[ cell ];
11         p /= lambda[ cell ] + alpha[ cell ] * pressure[ cell ];
12         pressure[ cell ] = p;
13     }
14     return true;
15 }
```

3.2 Implementace paralelního řešiče pomocí knihovny TNL

V této kapitole popíšeme implementaci paralelního řešiče s využitím knihovny TNL, který je hlavním výsledkem této práce. Díky návrhu knihovny TNL je možné spustit výsledný program jak na CPU, tak i masivně paralelně na GPU s využitím architektury CUDA. Při implementaci řešiče využíváme mnoho objektů z knihovny TNL poskytujících jednotné rozhraní pro práci s paměti na GPU a běžnou operační pamětí a pro výběr sériových nebo paralelních algoritmů v závislosti na tom, jestli je program spuštěn na CPU nebo na GPU. Při popisu se zaměřujeme na samotnou implementaci algoritmu 1, také však stručně popíšeme důležité objekty knihovny TNL použité při implementaci.

Implementujeme program `tnl-single-phase`, při výpočtu ale využíváme programy z knihovny TNL, např. `tnl-grid-setup` pro generování sítě a `tnl-init` pro generování diskretizované počáteční podmínky. Základní objekty použité při implementaci řešiče představíme pomocí algoritmu 3. Hlavní rozdíly oproti implementaci pomocí knihovny UMFPACK poté podrobněji popíšeme v následujících podkapitolách.

Algoritmus 3

1. Pomocí programu `tnl-grid-setup` vygenerujeme pro zvolené parametry $N_x, N_y \in \mathbb{N}$ numerickou síť $\mathcal{K}_h$ o rozměrech $N_x \times N_y$ pokrývající obdélníkovou oblast Ω, vygenerovaný objekt uložíme do souboru. Symboly h_x, h_y označíme rozměry elementů v příslušném směru a volíme parametr sítě $h = \min\{h_x, h_y\}$.

2. Pomocí programu `tnl-init` diskretizujeme na dané síti počáteční podmínku (1.8a), střední hodnoty tlaku na elementech p_K^0 reprezentujeme pomocí vektoru uloženého do souboru.

3. Dále v programu `tnl-single-phase` vytvoříme objekt `SinglePhaseSolver` reprezentující vlastní řešič a vstupní data charakterizující úlohu načteme jako parametry předané na příkazové řádce. Parametry jako např. Φ, $\frac{M}{RT}$ nebo μ předáváme hodnotou, numerickou síť a počáteční podmínku specifikujeme jako název souboru obsahující objekt vytvořený v kroku 1. resp. 2. pomocí externího programu. Okrajové podmínky diskretizované pomocí (2.46) resp. (2.45) reprezentujeme přímo v kódu řešiče pomocí objektu `SinglePhaseBoundaryConditions`.

4. Prvky lokálních matic $\mathbf{A}_K$ na jednotlivých elementech $K \in \mathcal{K}_h$ vypočteme dle (2.39) v rámci metody `SinglePhaseSolver::preIterate` (viz 3.2.2). Lokální vektory $\boldsymbol{G}_K$ definované dle (2.35) reprezentujeme pomocí metody `G_KE`.

5. Položíme $i = 0$.

6. Dokud $t_i < T_f$, opakujeme následující kroky:

   6.1. Pomocí metody `SinglePhaseSolver::preIterate` (viz 3.2.2) aktualizujeme prvky pomocných vektorů.

   6.2. V metodě `SinglePhaseSolver::assemblyLinearSystem` (viz 3.2.1) sestavíme matici soustavy a vektor pravé strany pro soustavu (2.52).

   6.3. Soustavu lineárních rovnic pro neznámé p_E^{i+1} z předchozího bodu řešíme pomocí iterační metody GMRES[4] [14, 13], jejíž implementace [12] je součástí knihovny `TNL`.

   6.4. Pomocí metody `SinglePhaseSolver::postIterate` (viz 3.2.3) s využitím (2.49) aktualizujeme střední hodnoty tlaku na elementech.

   6.5. Přejdeme na následující časovou hladinu, $i \to i + 1$.

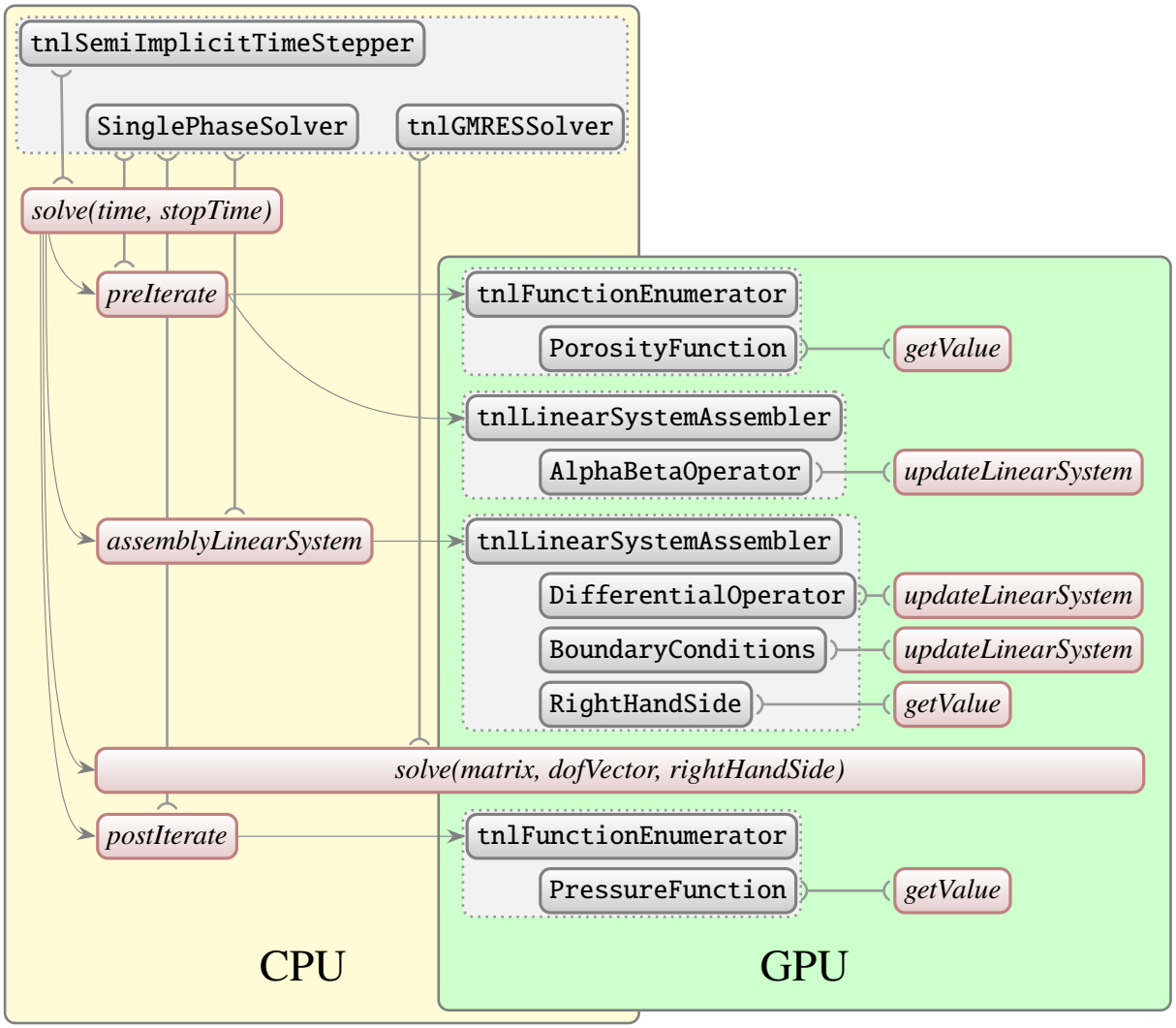
7. V čase $t_i = T_f$ získáme numerické řešení dané úlohy.

Veškerá vstupní data charakterizující úlohu, časovou a prostorovou diskretizaci, parametry řešiče GMRES apod. specifikujeme jako argumenty programu `tnl-single-phase` na příkazové řádce, o jejich načtení ve 3. kroku algoritmu 3 se starají série tříd s příponami `Config` a `Setter`. Jako numerickou síť používáme třídu `tnlGrid`, která ve dvoudimenzionálním případě reprezentuje ekvidistantní konformní obdélníkovou síť. Elementy sítě označujeme v kódu jako *cell* a hrany jako *face*, čemuž odpovídají také názvy proměnných v kódu. Hodnoty p_E^{i+1}, takzvané *stupně volnosti* (DOFs[5]) představující neznámé při řešení soustavy lineárních rovnic na i-té časové hladině v bodu 6.3., ukládáme pomocí objektu typu `tnlVector`. V mnoha objektech definujeme pomocný typ `DofVectorType` jako označení pro typ `tnlVector` se specifickými šablonovými parametry určujícími mj. umístění dat, což je buď standardní operační paměť nebo paměť GPU. Hlavní matici soustavy ukládáme pomocí objektu typu `tnlSlicedEllpackMatrix`, což je formát pro ukládání řídkých matic optimalizovaný pro GPU [11].

V 5. resp. 6. bodu algoritmu využijeme objekty typu `tnlPDESolver` a `tnlSemiImplicitTimeStepper` z knihovny `TNL`, které obecně popisují krokování v čase při řešení parciálních diferenciálních rovnic pomocí semi-implicitních schémat. Relevantní část algoritmu 3 je schematicky znázorněna na diagramu 3.3. O přechod mezi časovými hladinami se stará objekt typu `tnlSemiImplicitTimeStepper` resp. jeho metoda `solve` odpovídající 6. bodu algoritmu. V levé části diagramu jsou zobrazeny metody třídy `SinglePhaseSolver`, v pravé části pak třídy a jejich metody spojené se samotným výpočtem, které jsou podrobně popsány v následujících podkapitolách.

[4] Generalized Minimal Residual method

[5] Degrees of Freedom



Obrázek 3.3: Hlavní objekty a metody vystupující v řešiči. O řízení výpočtu se stará objekt typu `tnlSemiImplicitTimeStepper`, jehož metoda `solve` v každém časovém kroku volá jednotlivé metody odpovídající bodům 6.1. až 6.4. algoritmu 3. Žlutý obdélník obsahuje metody spouštěné na CPU, metody a třídy v zeleném obdélníku pracují přímo s GPU.

Z hlediska spouštění daného kódu lze běžné metody spouštět pouze na CPU, metody určené ke spouštění na GPU je nutné příslušně označit. V rámci `TNL` se používá označení `__cuda_callable__`, což umožní volání takto označených metod z kódu běžícího na GPU i z kódu běžícího na CPU. Na obr. 3.3 jsou metody odlišeny barevným pozadím, žlutý obdélník obsahuje metody spouštěné na CPU a zelený obdélník obsahuje metody spouštěné paralelně na GPU. V případě metody `solve` třídy `tnlGMRESSolver` spouštěné na CPU jsme pomocí přesahu přes obě barevná pozadí znázornili skutečnost, že značná část výpočtu probíhá pomocí dalších objektů na GPU.

Další objekty z knihovny `TNL` použité při implementaci popíšeme v následujících podkapitolách.

3.2.1 Sestavení matice soustavy

Nejprve popíšeme způsob sestavení hlavní matice soustavy pomocí metody `assemblyLinearSystem` třídy `SinglePhaseSolver`. Využijeme objekt typu `tnlLinearSystemAssembler`, který s využitím dalších objektů paralelně prochází danou numerickou síť a pomocí uživatelských objektů sestavuje matici a vektor pravé strany soustavy.

Existují dva hlavní přístupy k sestavování hlavní matice: první možností je procházet numerickou síť po elementech, na každém elementu K spočítat hodnoty $\widetilde{B}_{K,E,F}$ pro všechny hrany $E, F \in \mathcal{E}_K$ a tyto hodnoty přičíst ke všem relevantním prvkům matice soustavy. Tento postup však není vhodný z hlediska paralelizace, jelikož vlákna zpracovávající dva sousední elementy K_1, K_2 by nevyhnutelně zapisovala do stejné oblasti v paměti příslušející společné hraně $E \in \mathcal{E}_{K_1} \cap \mathcal{E}_{K_2}$. Proto použijeme druhý přístup, kdy síť procházíme po hranách a pro každou hranu E nastavíme příslušný řádek matice soustavy. Při tom využíváme toho, že každý řádek má jen malé množství nenulových prvků v závislé pouze na geometrii elementů sítě, což je jedna z hlavních výhod MHFEM. V případě obdélníkových elementů mají řádky odpovídající vnitřním hranám sítě 7 nenulových prvků, řádky odpovídající hranám s Neumannovou podmínkou mají 4 nenulové prvky a řádky odpovídající hranám s Dirichletovou podmínkou mají pouze 1 nenulový prvek na diagonále.

Při vytváření instance `systemAssembler` specifikujeme pomocí šablonových parametrů typy objektů, se kterými bude `systemAssembler` pracovat. Instance těchto tříd pak předáme jako parametry metody `assembly`. Soustava (2.52) získaná metodou hybridizace neodpovídá schématu implementovanému v `TNL`, proto pomocí typu `tnlNoTimeDiscretisation` specifikujeme chování objektu `systemAssembler` tak, aby se o sestavení matice staraly pouze uživatelské objekty.

```
1  tnlLinearSystemAssembler< MeshType, DofVectorType, DifferentialOperator, BoundaryConditions,
   ↪  RightHandSide, tnlNoTimeDiscretisation, MatrixType > systemAssembler;
2  systemAssembler.template assembly< MeshType::Dimensions - 1 >( time,
3                                                                 tau,
4                                                                 mesh,
5                                                                 differentialOperator,
6                                                                 boundaryConditions,
7                                                                 rightHandSide,
8                                                                 dofVector,
9                                                                 matrix,
10                                                                b );
```

Šablonový parametr `MeshType::Dimensions - 1` specifikuje, že se numerická síť bude procházet po hranách. Parametry `time` a `tau` typu `RealType` určují časovou hladinu t_{i+1} resp. časový krok Δt_i, numerická síť je reprezentována objektem `mesh`. Následující tři parametry specifikují objekty, které se starají o samotné výpočty. Objekt `differentialOperator` nastavuje řádky matice odpovídající vnitřním hranám a příspěvek vektoru $\boldsymbol{G}_K$ na vnitřních hranách k pravé straně, zbylé příspěvky k vektoru pravé strany počítá objekt `rightHandSide`. O nastavení řádků odpovídajících okrajovým hranám se stará objekt `boundaryConditions`. Poslední tři parametry představují vektor nezávisle proměnných `dofVector`, matici soustavy `matrix` a vektor pravé strany `b`.

V následujících podkapitolách popíšeme objekty diferenciální operátor, pravou stranu a okrajové podmínky, sloužící k sestavování hlavní matice. Při výpočtu prvků matice podobně jako v kapitole 3.1.4 používáme pomocné objekty `alpha`, `beta` a `lambda`, které blíže popíšeme v kapitole 3.2.2.

3.2.1.1 Diferenciální operátor

Objekt `differentialOperator` šablonového typu `DifferentialOperator` v předchozí sekci představuje instanci třídy `SinglePhaseOperator` popisující sestavení části matice soustavy (2.52) odpovídající

(a) Elementy se společnou vertikální hranou.

(b) Elementy se společnou horizontální hranou.

Obrázek 3.4: Očíslování hran sousedních elementů $0, 1$ (vyznačeny černě) se společnou hranou.

vnitřním hranám sítě. Hlavní metoda třídy `SinglePhaseOperator` je `updateLinearSystem`, ve které nastavíme řádek matice soustavy příslušný vnitřní hraně sítě a příspěvek gravitace přičteme k vektoru pravé strany. Šablonové parametry, které nejsou důležité pro následující výklad, pro přehlednost skryjeme pomocí trojtečky (`...`).

```
template< ... >
void SinglePhaseOperator< ... >::updateLinearSystem( const RealType & time,
                                                     const RealType & tau,
                                                     const MeshType & mesh,
                                                     const IndexType & indexRow,
                                                     const CoordinatesType & coordinates,
                                                     Vector & u,
                                                     Vector & b,
                                                     MatrixRow & matrixRow ) const
{
    // contribution to the right hand side
    RealType bValue = 0.0;

    // indexes of the right/top (cellIndexes[0]) and left/bottom (cellIndexes[1]) cells
    IndexType cellIndexes[ 2 ];
    getCellsForFace( mesh, indexRow, cellIndexes );

    // indexes of the faces, sorted according the following diagrams
    IndexType faceIndexes[ 8 ];
```

Pro daný řádek s indexem `indexRow` nastavujeme celkem 7 nenulových prvků odpovídajících hranám sousedních elementů se společnou hranou `indexRow`. S využitím pomocné funkce `getCellsForFace` uložíme do pole `cellIndexes` indexy sousedních elementů hrany `indexRow`, přičemž index `cellInxexes[0]` odpovídá pravému resp. hornímu elementu a index `cellIndexes[1]` odpovídá levému resp. dolnímu elementu v případě vertikální resp. horizontální hrany `indexRow`.

Dále alokujeme pole `faceIndexes` pro indexy hran, které získáme analogicky pomocí funkce `getFacesForCell`. Abychom zachovali uspořádání nenulových prvků na daném řádku podle indexů hran, musíme rozlišit dva případy podle orientace hrany `indexRow`. V rámci objektu `tnlGrid` jsou nejprve číslovány vertikální hrany a poté horizontální hrany, tedy opačně než bylo popsáno v kapitole 3.1.1. Číslování hran v obou případech ze zobrazeno na obr. 3.4.

V případě vertikální hrany seřadíme hrany dle obr. 3.4a a pro výpočet hodnot nenulových prvků použijeme pomocnou metodu `getValue`, kterou popíšeme později. Pro společnou hranu `indexRow` (odpovídající lokálním indexům 1 a 2) je třeba sečíst příspěvky na obou elementech.

```
    if( isVerticalFace( mesh, indexRow ) ) {
        getFacesForCell( mesh, cellIndexes[ 1 ],
            faceIndexes[ 0 ], faceIndexes[ 1 ], faceIndexes[ 4 ], faceIndexes[ 6 ] );
        getFacesForCell( mesh, cellIndexes[ 0 ],
            faceIndexes[ 2 ], faceIndexes[ 3 ], faceIndexes[ 5 ], faceIndexes[ 7 ] );

        matrixRow.setElement( 0, faceIndexes[ 0 ],
            getValue( mesh, bValue,  indexRow, faceIndexes[ 0 ], cellIndexes[ 1 ] ) );
        matrixRow.setElement( 1, faceIndexes[ 1 ],
            getValue( mesh, bValue,  indexRow, faceIndexes[ 1 ], cellIndexes[ 1 ] ) +
            getValue( mesh, bValue,  indexRow, faceIndexes[ 2 ], cellIndexes[ 0 ] ) );
        matrixRow.setElement( 2, faceIndexes[ 3 ],
            getValue( mesh, bValue,  indexRow, faceIndexes[ 3 ], cellIndexes[ 0 ] ) );
        matrixRow.setElement( 3, faceIndexes[ 4 ],
            getValue( mesh, bValue,  indexRow, faceIndexes[ 4 ], cellIndexes[ 1 ] ) );
        matrixRow.setElement( 4, faceIndexes[ 5 ],
            getValue( mesh, bValue,  indexRow, faceIndexes[ 5 ], cellIndexes[ 0 ] ) );
        matrixRow.setElement( 5, faceIndexes[ 6 ],
            getValue( mesh, bValue,  indexRow, faceIndexes[ 6 ], cellIndexes[ 1 ] ) );
        matrixRow.setElement( 6, faceIndexes[ 7 ],
            getValue( mesh, bValue,  indexRow, faceIndexes[ 7 ], cellIndexes[ 0 ] ) );
    }
```

V případě horizontální hrany seřadíme hrany dle obr. 3.4b a dále postupujeme analogicky jako v případě vertikální hrany. Na závěr přičteme příspěvek k vektoru pravé strany.

```
    else {
        getFacesForCell( mesh, cellIndexes[ 1 ],
            faceIndexes[ 0 ], faceIndexes[ 1 ], faceIndexes[ 4 ], faceIndexes[ 5 ] );
        getFacesForCell( mesh, cellIndexes[ 0 ],
            faceIndexes[ 2 ], faceIndexes[ 3 ], faceIndexes[ 6 ], faceIndexes[ 7 ] );

        matrixRow.setElement( 0, faceIndexes[ 0 ],
            getValue( mesh, bValue,  indexRow, faceIndexes[ 0 ], cellIndexes[ 1 ] ) );
        matrixRow.setElement( 1, faceIndexes[ 1 ],
            getValue( mesh, bValue,  indexRow, faceIndexes[ 1 ], cellIndexes[ 1 ] ) );
        matrixRow.setElement( 2, faceIndexes[ 2 ],
            getValue( mesh, bValue,  indexRow, faceIndexes[ 2 ], cellIndexes[ 0 ] ) );
        matrixRow.setElement( 3, faceIndexes[ 3 ],
            getValue( mesh, bValue,  indexRow, faceIndexes[ 3 ], cellIndexes[ 0 ] ) );
        matrixRow.setElement( 4, faceIndexes[ 4 ],
            getValue( mesh, bValue,  indexRow, faceIndexes[ 4 ], cellIndexes[ 1 ] ) );
        matrixRow.setElement( 5, faceIndexes[ 5 ],
            getValue( mesh, bValue,  indexRow, faceIndexes[ 5 ], cellIndexes[ 0 ] ) +
            getValue( mesh, bValue,  indexRow, faceIndexes[ 6 ], cellIndexes[ 1 ] ) );
        matrixRow.setElement( 6, faceIndexes[ 7 ],
            getValue( mesh, bValue,  indexRow, faceIndexes[ 7 ], cellIndexes[ 0 ] ) );
    }

    b[ indexRow ] += bValue;
}
```

Pro výpočet nenulových prvků matice jsme použili metodu `getValue`, která vrací hodnoty koeficientů $\widetilde{B}_{K,E,F}$ vystupujících v rovnici (2.52) a do proměnné `bValue` přičte gravitační příspěvek $\widetilde{B}_{K,E,F}G_{K,F}$.

```
template< ... >
RealType SinglePhaseOperator< ... >::getValue( const MeshType & mesh,
                                               RealType & bValue,
                                               const IndexType & indexRow,
                                               const IndexType & indexColumn,
                                               const IndexType & indexCell ) const
{
    RealType aux1 = beta->getElementFast( indexCell, indexRow );
    RealType aux2 = beta->getElementFast( indexCell, indexColumn );
    RealType value = - aux1 * pressure[ indexCell ] * aux2 * pressure[ indexCell ] /
                ( lambda[ indexCell ] + alpha[ indexCell ] * pressure[ indexCell ] );
    if( indexRow == indexColumn )
        value += beta->getElementFast( indexCell, indexColumn ) * pressure[ indexCell ];
    bValue += value * G_KE( mesh, indexCell, indexColumn ) * pressure[ indexCell ];
    return value;
}
```

3.2.1.2 Pravá strana

Složky vektoru pravé strany odpovídající vnitřním hranám sítě se nastavují pomocí objektu `rightHandSide` šablonového typu `RightHandSide`, který v našem řešiči podobně jako v případě diferenciálního operátoru představuje instanci třídy `SinglePhaseRightHandSide`. Výpočet odpovídající pravé straně rovnice (2.52a) provedeme v metodě `getValue`.

```
template< ... >
RealType SinglePhaseRightHandSide< ... >::getValue( const MeshType & mesh,
                                                    const IndexType & indexRow,
                                                    const RealType & time ) const
{
    IndexType cellIndexes[ 2 ];
    getCellsForFace( mesh, indexRow, cellIndexes );

    RealType result = 0.0;
    for( IndexType i = 0; i < numCells; i++ ) {
        const IndexType & cell = cellIndexes[ i ];
        result += beta->getElementFast( cell, indexRow ) * pressure[ cell ] /
                    ( lambda[ cell ] + alpha[ cell ] * pressure[ cell ] ) *
                    ( F[ cell ] + lambda[ cell ] * pressure[ cell ] );
    }
    return result;
}
```

3.2.1.3 Okrajové podmínky

Objekt `boundaryConditions` představuje instanci třídy `SinglePhaseBoundaryConditions` popisující část lineárního systému dle rovnic (2.52b) a (2.52c). Hlavní metoda této třídy je podobně jako v případě diferenciálního operátoru `updateLinearSystem`. Index sousedního elementu pro danou okrajovou hranu získáme stejně jako v případě vnitřní hrany pomocí funkce `getCellsForFace`.

```
template< ... >
void SinglePhaseBoundaryConditions< ... >::updateLinearSystem(
                                                const RealType & time,
                                                const MeshType & mesh,
                                                const IndexType & indexRow,
                                                const CoordinatesType & coordinates,
                                                DofVectorType & u,
                                                DofVectorType & b,
                                                MatrixRow & matrixRow ) const
{
    // only the first index is valid for boundary faces
    IndexType cellIndexes[ 2 ];
    getCellsForFace( mesh, indexRow, cellIndexes );
    const IndexType & indexCell = cellIndexes[ 0 ];
```

Dále odlišíme výpočet podle typu okrajové podmínky na dané hraně. V případě Dirichletovy okrajové podmínky stačí dle (2.52c) stačí na diagonálu matice zapsat hodnotu 1, příspěvek okrajové podmínky p_{Dir} diskretizované dle (2.46) počítáme pomocí metody `getDirichletValue`.

```
    if( isDirichletBoundary( mesh, indexRow ) ) {
        matrixRow.setElement( 0, indexRow, 1.0 );
        b[ indexRow ] = getDirichletValue( mesh, indexRow, time );
    }
```

V případě Neumannovy okrajové podmínky provádíme výpočet dle (2.52b) podobně jako v případech diferenciálního operátoru a pravé strany. Příspěvek okrajové podmínky q_{Neu} diskretizované dle (2.45) počítáme pomocí metody `getNeumannValue`.

```
    else {
        // prepare face indexes
        IndexType faceIndexes[ 4 ];
        getFacesForCell( mesh, indexCell,
            faceIndexes[ 0 ], faceIndexes[ 1 ], faceIndexes[ 2 ], faceIndexes[ 3 ] );
        // prepare right hand side value
        RealType bValue = getNeumannValue( mesh, indexRow, time ) +
                    beta->getElementFast( indexCell, indexRow ) * pressure[ indexCell ] /
                    ( lambda[ indexCell ] + alpha[ indexCell ] * pressure[ indexCell ] ) *
                    ( F[ indexCell ] + lambda[ indexCell ] * pressure[ indexCell ] );
        // set matrix elements, add to bValue
        matrixRow.setElement( 0, faceIndexes[ 0 ],
            getValue( mesh, bValue,  indexRow, faceIndexes[ 0 ], indexCell ) );
        matrixRow.setElement( 1, faceIndexes[ 1 ],
            getValue( mesh, bValue,  indexRow, faceIndexes[ 1 ], indexCell ) );
        matrixRow.setElement( 2, faceIndexes[ 2 ],
            getValue( mesh, bValue,  indexRow, faceIndexes[ 2 ], indexCell ) );
        matrixRow.setElement( 3, faceIndexes[ 3 ],
            getValue( mesh, bValue,  indexRow, faceIndexes[ 3 ], indexCell ) );
        // set right hand side
        b[ indexRow ] = bValue;
    }
}
```

3.2.2 Výpočet pomocných vektorů

Při sestavování matice soustavy (2.52) v kapitole 3.2.1 na mnoha místech používáme hodnoty $\widetilde{A}^i_{K,E}$ a $\widetilde{A}^i_K$ definované pomocí (2.50). Abychom se vyhnuli zbytečnému opakování výpočtů, využijeme pomocné datové struktury a uložíme hodnoty těchto koeficientů do paměti. Pro tyto účely použijeme metodu `preIterate`.

Nejprve vypočteme hodnoty koeficientů λ^i_K definovaných dle (2.47). Pro paralelizaci výpočtu využijeme objekt `tnlFunctionEnumerator`, který vyčíslí hodnoty síťové funkce `porosityFunction` na síti `mesh`, vynásobí je konstantou odpovídající výrazu $\frac{M|K|}{RT\Delta t_i}$ a výsledek uloží do vektoru `lambda`. Typy objektů specifikujeme pomocí šablonových parametrů třídy, dimenzi síťových entit stanovíme pomocí šablonového parametru metody `enumerate`.

```
PorosityFunction< MeshType, RealType, IndexType > porosityFunction;
porosityFunction.bind( porosity );
tnlFunctionEnumerator< MeshType,
                       PorosityFunction< MeshType, RealType, IndexType >,
                       DofVectorType > lambdaEnumerator;
lambdaEnumerator.template enumerate< MeshType::Dimensions >(
                                 mesh,
                                 porosityFunction,
                                 lambda,
                                 idealGasCoefficient * mesh.getHxHy() / tau );
```

Dále podobně jako v kapitole 3.1.3 rozložíme koeficienty $\widetilde{A}^i_K$ a $\widetilde{A}^i_{K,E}$ na součin $\widetilde{A}^i_K = p^i_K \alpha_K$ a $\widetilde{A}^i_{K,E} = p^i_K \beta_{K,E}$, kde α_K a $\beta_{K,E}$ jsou konstantní v čase, výpočet tedy provedeme pouze pro $t = 0$. Pro uložení koeficientů α_K stačí použít vektor, hodnoty $\beta_{K,E}$ uložíme pomocí řídké matice indexované pomocí řádkového indexu K a sloupcového indexu E. Pro sestavení pomocné matice `beta` a vektoru `alpha` zneužijeme objekt `tnlLinearSystemAssembler`, který jsme již použili v kapitole 3.2.1 pro sestavení hlavní matice soustavy. Hodnoty koeficientů vypočteme pomocí „operátoru“ `AlphaBetaOperator`, a to pro vnitřní i okrajové entity,[6] jako „pravou stranu“ použijeme nulovou funkci a pomocí tagu `tnlNoTimeDiscretization` vyloučíme vnitřní výpočty. Jelikož řádky matice `beta` odpovídají elementům sítě, procházíme síť po elementech.

```
AlphaBetaOperator< MeshType, RealType, IndexType > alphaBetaOperator( idealGasCoefficient,
 ↪  permeability, viscosity );
tnlConstantFunction< MeshType::Dimensions, RealType > nullFunction;
tnlLinearSystemAssembler< MeshType,
                          DofVectorType,
                          AlphaBetaOperator< MeshType, RealType, IndexType >,
                          AlphaBetaOperator< MeshType, RealType, IndexType >,
                          tnlConstantFunction< MeshType::Dimensions, RealType >,
                          tnlNoTimeDiscretisation,
                          MatrixType > systemAssembler;
```

[6] V třídě `AlphaBetaOperator` musíme definovat metody `updateLinearSystem` pro vnitřní a okrajové síťové entity, které mají různé signatury. Jelikož v obou případech provádíme stejný výpočet, definujeme privátní metodu `update`, kterou voláme z `updateLinearSystem`.

```
systemAssembler.template assembly< MeshType::Dimensions >(
                                time,
                                tau,
                                mesh,
                                alphaBetaOperator,   // inner cells
                                alphaBetaOperator,   // boundary cells
                                nullFunction,        // right hand side
                                auxiliaryDofVector,  // pressure as DofVectorType
                                beta,    // matrix
                                alpha ); // right hand side
```

Implementace hlavních metod objektů `PressureFunction` a `AlphaBetaOperator` jsou triviální a jelikož definice tříd obsahují více metakódu než relevantní informace, konkrétní kód zde neuvádíme.

3.2.3 Výpočet tlaku na elementech

V metodě `postIterate` aktualizujeme střední hodnoty tlaku na elementech dle vztahu (2.49). Pro paralelizaci výpočtu využijeme již známý objekt `tnlFunctionEnumerator`, který vyčíslí hodnoty síťové funkce `pressureFunction` na síti `mesh` a výsledek uloží do vektoru `auxiliaryDofVector`. Dimenzi síťových entit opět specifikujeme pomocí šablonového parametru metody `enumerate`. Jelikož hodnoty p_K^{i+1} dle (2.49) nezávisí na tlaku na sousedních elementech, ale pouze na hodnotě p_K^i z předchozí časové vrstvy, nedojde během výpočtu ke konfliktu při přístupu k vektorům `pressure` a `auxiliaryDofVector`, které odkazují na stejné místo v paměti.

```
PressureFunction< MeshType, RealType, IndexType > pressureFunction( idealGasCoefficient,
 ↪  grav_y );
pressureFunction.bind( alpha, *betaDevice, lambda, pressure, ptrace, F );
tnlFunctionEnumerator< MeshType,
                       PressureFunction< MeshType, RealType, IndexType >,
                       DofVectorType > pressureEnumerator;
pressureEnumerator.template enumerate< MeshType::Dimensions >( mesh,
                                                               pressureFunction,
                                                               auxiliaryDofVector );
```

Třída `PorosityFunction` představuje síťovou funkci, jejíž rozhraní spočívá v implementaci metody `getValue`. Pro výpočet dle (2.49) potřebujeme znát indexy sousedních hran daného elementu, k čemuž použijeme funkci `getFacesForCell`. Dále postupně v proměnné `result` počítáme požadovanou hodnotu.

```
template< ... >
RealType PressureFunction::getValue( const MeshType & mesh,
                                     const IndexType & indexCell,
                                     const RealType & time ) const
{
    IndexType faceIndexes[ 4 ];
    getFacesForCell( mesh, indexCell, faceIndexes[ 0 ], faceIndexes[ 1 ], faceIndexes[ 2 ],
     ↪  faceIndexes[ 3 ] );
    RealType result = 0.0;
    for( int i = 0; i < 4; i++ ) {
        const IndexType & indexFace = faceIndexes[ i ];
        result += beta->getElementFast( indexCell, indexFace ) * ( ptrace[ indexFace ] -
         ↪  G_KE( mesh, indexCell, indexFace ) * pressure[ indexCell ] );
    }
    result *= pressure[ indexCell ];
    result += F[ indexCell ] + lambda[ indexCell ] * pressure[ indexCell ];
    result /= lambda[ indexCell ] + alpha[ indexCell ] * pressure[ indexCell ];
    return result;
}
```

Kapitola 4

Numerické výsledky

Poslední kapitola této práce se zabývá výsledky numerických simulací. Hlavním cílem této kapitoly je ověřit konvergenci numerického schématu pro řešení soustavy (1.7) odvozeného v kapitole 2 a srovnat jednotlivé implementace řešičů popsané v kapitole 3, a to z hlediska přesnosti a efektivity.

Nejprve využijeme speciální případ se známým analytickým řešením pro ověření konvergence numerického schématu pomocí experimentální analýzy konvergence a pro porovnání přesnosti řešičů. Kromě základního numerického schématu zkoumáme také modifikované schéma s upwindovou stabilizací popsanou v kapitole 2.6. Dále na úloze stlačitelného proudění bez gravitace blíže zkoumáme efektivitu `TNL` řešiče z hlediska výpočetních časů při použití různých CPU a GPU. V advekční úloze stlačitelného proudění ovlivněného gravitací se zabýváme stabilitou numerického schématu bez a s upwindovou stabilizací.

Všechny výsledky prezentované v této kapitole byly vypočteny pomocí vlastních řešičů popsaných v kapitole 3. Pro zpracování výsledků jsme použili program GNU Parallel [15] a program Gnuplot [16] pro vizualizaci výsledků.

4.1 Barenblattova úloha

V této kapitole se budeme zabývat speciálním případem soustavy (1.7), pro který je známé analytické řešení. To nám umožní ověříme konvergenci numerického schématu a porovnat přesnost jednotlivých implementací.

Volbou parametrů $\Phi = 1$, $f = 0$, $\boldsymbol{g} = 0$, $\mathbf{K} = 1$, $\mu = \frac{1}{2}$ a $M = RT$ přejde soustava (1.7) do tvaru

$$\frac{\partial p}{\partial t} + \nabla \cdot (p\boldsymbol{u}) = 0, \tag{4.1a}$$

$$\boldsymbol{u} = -2\nabla p. \tag{4.1b}$$

Dále předepíšeme počáteční a okrajové podmínky ve tvaru

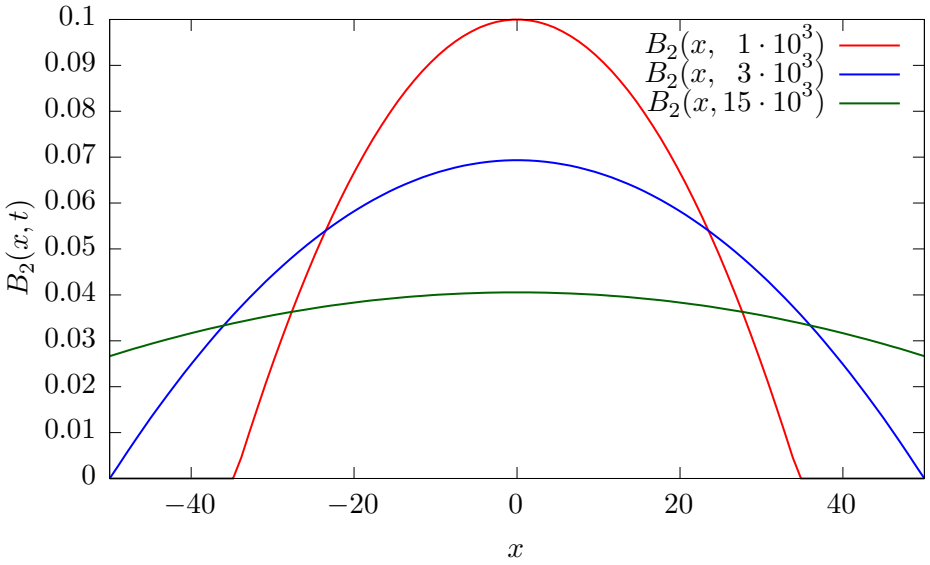
$$p(\boldsymbol{x}, t_0) = B_2(x, t_0), \qquad \forall \boldsymbol{x} \in \Omega, \tag{4.2a}$$

$$p(\boldsymbol{x}, t) = B_2(x, t), \qquad \forall \boldsymbol{x} \in \Gamma_{Dir}, t \in (t_0, T_f), \tag{4.2b}$$

$$\boldsymbol{q}(\boldsymbol{x}, t) \cdot n(\boldsymbol{x}) = 0, \qquad \forall \boldsymbol{x} \in \Gamma_{Neu}, t \in (t_0, T_f), \tag{4.2c}$$

kde $B_2(x, t)$ je Barenblattovo řešení rovnice (4.1) dané vztahem

$$B_m(x, t) = t^{-k} \left[\left(1 - \frac{k(m-1)}{2m} \frac{|x|^2}{t^{2k}} \right)_+ \right]^{\frac{1}{m-1}}, \qquad m = 2, \tag{4.3}$$



Obrázek 4.1: Graf Barenblattova řešení $B_2(x,t)$ na intervalu $[-50,50]$ pro hodnoty parametru $t = 1\cdot 10^3, 3\cdot 10^3, 15\cdot 10^3$.

kde $\xi_+ = \max(\xi, 0)$ a $k = (m+1)^{-1}$. Řešení (4.3) má v každém čase $t > 0$ kompaktní nosič $[-\alpha_m(t), \alpha_m(t)]$ rozšiřující se konečnou rychlostí [18], kde

$$\alpha_m(t) = \sqrt{\frac{2m}{k(m-1)}} \cdot t^k. \tag{4.4}$$

Na obr. 4.1 ze zobrazen graf Barenblattova řešení $B_2(x,t)$ na intervalu $[-50,50]$ pro různé hodnoty parametru t.

4.1.1 Experimentální analýza konvergence

Numerické řešení úlohy (4.1) srovnáme s jejím analytickým řešením (4.3) pomocí experimentálních řádů konvergence (EOC[1]). Přesnost časové diskretizace (2.43) je prvního řádu, dále zkoumáme řád přesnosti prostorové diskretizace v normách L^1, L^2 a L^∞. Předpokládáme, že chyba E_h numerického řešení na síti $\mathcal{K}_h$ má v normě L^ν tvar

$$\|E_h\|_\nu = C_1\Delta t + C_2 h^{\alpha_\nu}, \tag{4.5}$$

kde C_1, C_2 jsou konstanty a $\Delta t = \max_{i\in\widehat{n}} \Delta t_i$ je nejdelší časový krok při diskretizaci $\mathcal{T}$. Řád konvergence α_ν lze určit experimentálně pomocí analýzy numerických řešení na sítích s různými parametry h, přičemž pro omezení chyby časové diskretizace zmenšujeme časový krok úměrně jisté mocnině parametru sítě h, $\Delta t \sim h^\beta$. Pro $\beta < \alpha_\nu$ bude při zjemňování sítě chyba numerického řešení $\|E_h\|_\nu$ dána převážně chybou časové diskretizace, pro $\beta > \alpha_\nu$ naopak chybou prostorové diskretizace. Pro chyby numerických řešení $\|E_{h_1}\|_\nu, \|E_{h_2}\|_\nu$ na sítích s parametry h_1 resp. h_2 definujeme experimentální řád konvergence v normě L^ν pomocí

$$\mathrm{EOC}_\nu = \frac{\ln\|E_{h_1}\|_\nu - \ln\|E_{h_2}\|_\nu}{\ln h_1 - \ln h_2}. \tag{4.6}$$

[1]Experimental Order of Convergence

Při určování chyby numerického řešení E_h vyčíslujeme hodnoty analytického řešení ve středech elementů sítě příslušného numerického řešení.

Úlohu (4.1) s podmínkami (4.2) řešíme pro počáteční čas $t_0 = 3 \cdot 10^3$ s a koncový čas $T_f = 15 \cdot 10^3$ s na obdélníkové oblasti Ω o rozměrech $100 \times 100\,\mathrm{m}^2$, přičemž Dirichletova část hranice Γ_{Dir} je tvořena vertikálními stranami $x = -50$ a $x = 50$ ($0 \leq y \leq 100$) a horizontální strany $y = 0$ a $y = 100$ ($-50 \leq x \leq 50$) tvoří Neumannovu část hranice Γ_{Neu}. Numerické řešení spočítáme na sítích o rozměrech $N_x \times N_y$ elementů, kde $N_x = N_y$ a při zjemňování sítě volíme $N_x = 25$, $N_x = 50$, $N_x = 100$, $N_x = 200$ a $N_x = 400$, tedy na zkoumané oblasti $h = h_x = h_y = \frac{100}{N_x}$. Časový krok volíme $\Delta t = 1{,}25\,\mathrm{s}$ na nejjemnější síti $N_x = 400$ a při každém přechodu na hrubší síť je časový krok $4\times$ větší oproti předchozí síti, tedy $\Delta t \sim h^2$.

Tab. 4.1 obsahuje EOC a chyby numerických řešení popsané úlohy při použití `UMFPACK` řešiče. Při použití `TNL` řešiče se soustavy lineárních rovnic řeší pomocí iterační metody GMRES a výsledky tak závisí na zastavovacím kritériu.[2] Pro dosažení výsledků v tab. 4.2 jsme zvolili zastavovací kritérium $\varepsilon = 10^{-9}$ na základě analýzy chování chyb a EOC v závislosti na tomto parametru. Podle grafu na obr. 4.2 vidíme, že pro hrubé sítě jsou chyby a EOC téměř konstantní, ale pro jemné sítě je potřeba zvolit dostatečně malé ε, abychom dostali dostatečně přesné aproximace řešení a dobré řády konvergence. Volba $\varepsilon = 10^{-9}$ tedy odpovídá hodnotě, kdy se pro síť $N_x = 400$ EOC přiblíží asymptotické hodnotě. Jelikož knihovna `TNL` používá metodu GMRES s restartováním [13, str. 167], dalším důležitým parametrem je tedy počet iterací k restartování metody, který volíme $m = 10$. Dále vynucujeme minimální počet iterací GMRES metody $m_0 = 2$, abychom se vyhnuli některým nežádoucím jevům v důsledku nemonotonní konvergence metody. V praxi však při daném nastavení metoda GMRES konverguje průměrně po 30-50 iteracích.

Podle obou tabulek 4.1 a 4.2 pro jednotlivé řešiče jsou hodnoty EOC při použití numerického schématu bez upwindové stabilizace a při zmenšování časového kroku $\Delta t \sim h^2$ rovny přibližně 2. Při zmenšování časového kroku $\Delta t \sim h$ byly pozorovány EOC přibližně rovny 1, a tedy převážila chyba časové diskretizace. Naopak při zmenšování časového kroku s mocninou $\sigma > 2$, $\Delta t \sim h^\sigma$, zůstaly hodnoty EOC přibližně rovny 2. Lze tedy říci, že konvergence numerického schématu bez upwindové stabilizace je druhého řádu.

Rozdíly mezi řešiči z hlediska přesnosti jsou dány způsobem řešení soustavy lineárních rovnic (2.52). Přesnost řešiče využívajícího knihovnu `UMFPACK` je dle tab. 4.1 a 4.2 pro danou úlohu znatelně horší než při použití knihovny `TNL`, což lze přisoudit samoopravovací schopnosti iteračního řešiče. Při použití `TNL` řešiče se řešení získaná výpočtem na CPU a GPU mírně liší, což je způsobeno použitím paralelních algoritmů např. pro výpočet skalárních součinů a pro $\varepsilon \to 0$ se také tento rozdíl blíží k 0.

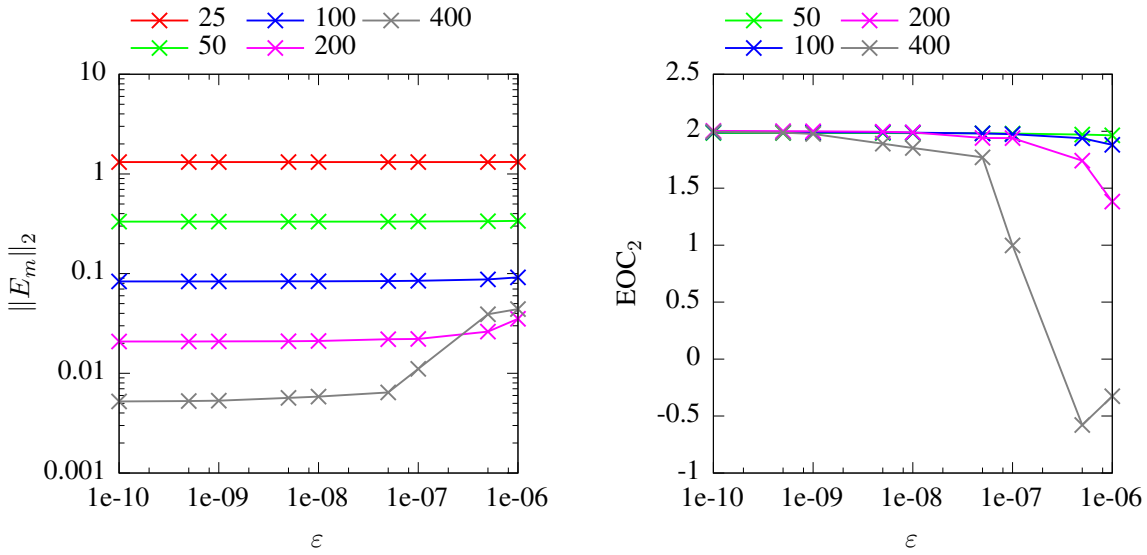
[2] Při řešení soustavy $\mathbf{M}\boldsymbol{v} = \boldsymbol{b}$ pomocí iterační metody konstruujeme posloupnost vektorů $\boldsymbol{v}^{(k)}$ konvergující k přesnému řešení $\boldsymbol{v}$. Metoda končí, pokud $\frac{\|\mathbf{M}\boldsymbol{v}^{(k)} - \boldsymbol{b}\|}{\|\boldsymbol{b}\|} < \varepsilon$, kde ε je zvolené zastavovací kritérium.

N_x	h	$\|E_h\|_1$	EOC_1	$\|E_h\|_2$	EOC_2	$\|E_h\|_\infty$	EOC_∞
25	4	$2{,}188 \cdot 10^{+00}$		$2{,}392 \cdot 10^{-02}$		$3{,}181 \cdot 10^{-04}$	
50	2	$5{,}638 \cdot 10^{-01}$	1,956	$6{,}152 \cdot 10^{-03}$	1,959	$8{,}178 \cdot 10^{-05}$	1,959
100	1	$1{,}438 \cdot 10^{-01}$	1,971	$1{,}568 \cdot 10^{-03}$	1,972	$2{,}084 \cdot 10^{-05}$	1,972
200	0,5	$3{,}600 \cdot 10^{-02}$	1,998	$3{,}926 \cdot 10^{-04}$	1,998	$5{,}219 \cdot 10^{-06}$	1,998
400	0,25	$9{,}011 \cdot 10^{-03}$	1,998	$9{,}825 \cdot 10^{-05}$	1,998	$1{,}306 \cdot 10^{-06}$	1,998

Tabulka 4.1: EOC v normách L^1, L^2 a L^∞ pro `UMFPACK` řešič na sítích o $N_x \times N_x$ elementech. Časový krok je volen $\Delta t = 1{,}25\,\text{s}$ na nejjemnější síti $N_x = 400$ a pro hrubší sítě $\Delta t \sim h^2$.

N_x	h	$\|E_h\|_1$	EOC_1	$\|E_h\|_2$	EOC_2	$\|E_h\|_\infty$	EOC_∞
25	4	$1{,}314 \cdot 10^{+00}$		$1{,}476 \cdot 10^{-02}$		$2{,}082 \cdot 10^{-04}$	
50	2	$3{,}322 \cdot 10^{-01}$	1,984	$3{,}729 \cdot 10^{-03}$	1,985	$5{,}242 \cdot 10^{-05}$	1,990
100	1	$8{,}361 \cdot 10^{-02}$	1,990	$9{,}373 \cdot 10^{-04}$	1,992	$1{,}318 \cdot 10^{-05}$	1,992
200	0,5	$2{,}090 \cdot 10^{-02}$	2,000	$2{,}343 \cdot 10^{-04}$	2,000	$3{,}296 \cdot 10^{-06}$	1,999
400	0,25	$5{,}311 \cdot 10^{-03}$	1,976	$5{,}935 \cdot 10^{-05}$	1,981	$8{,}487 \cdot 10^{-07}$	1,957

Tabulka 4.2: EOC v normách L^1, L^2 a L^∞ pro `TNL` řešič na sítích o $N_x \times N_x$ elementech. Časový krok je volen $\Delta t = 1{,}25\,\text{s}$ na nejjemnější síti $N_x = 400$ a pro hrubší sítě $\Delta t \sim h^2$. Parametr ε pro zastavení iteračního řešiče soustav lineárních rovnic volíme $\varepsilon = 10^{-9}$.



Obrázek 4.2: Závislost chyb numerických řešení a experimentálních řádů konvergence v L^2 normě na parametru ε pro zastavení iteračního řešiče soustav lineárních rovnic.

4.1.2 Efektivita výpočtu

Úloha se známým analytickým řešením také umožňuje srovnat efektivitu řešičů při nastavení vedoucímu k přibližně stejně přesným výsledkům. Pro porovnání efektivity využijeme úlohu popsanou v předchozí kapitole s identickým nastavením řešičů, včetně parametru $\varepsilon = 10^{-9}$ pro `TNL` řešič. Pro

výpočty jsme použili počítač označený v tab. 4.6 jako A, CPU_A pro sériové výpočty a GPU_A1 pro paralelní výpočty.

Tab. 4.3 obsahuje doby výpočtu numerického řešení při použití různých řešičů na jednotlivých sítích o $N_x \times N_x$ elementech. Jelikož nejvíce času při výpočtu zabírá řešení soustav lineárních rovnic, uvádíme ve druhém sloupci tab. 4.3 také počet hran sítě N_E (hlavní matice soustavy má rozměry $N_E \times N_E$). Pro danou úlohu je `TNL` řešič na CPU $4\times$ až $7\times$ rychlejší než `UMFPACK` řešič, přičemž chyba `TNL` řešiče je dle tab. 4.1 a 4.2 dokonce menší než chyba `UMFPACK` řešiče. Další výrazné urychlení je možné spuštěním `TNL` řešiče na GPU, pro nejjemnější síť $N_x = 800$ je paralelní výpočet více než $10\times$ rychlejší než výpočet na CPU. Na síti $N_x = 400$ je urychlení `TNL`-GPU řešiče téměř $50\times$ oproti `UMFPACK` řešiči.

N_x	N_E (DOFs)	Čas [s] (`UMFPACK`)	Urychlení $[-]$ (`UMFPACK` vs. `TNL` CPU)	Čas [s] (`TNL` CPU)	Urychlení $[-]$ (`TNL` CPU vs. `TNL` GPU)	Čas [s] (`TNL` GPU)
25	1300	0,6	8,1	0,1	0,1	0,7
50	5100	2,2	2,6	0,9	0,4	2,2
100	20200	41,3	3,5	11,9	1,4	8,4
200	80400	790,6	4,4	179,8	4,2	42,5
400	320800	18 288,0	7,1	2 560,2	7,1	362,5
800	1281600			44 081,2	10,7	4 105,4

Tabulka 4.3: Doby výpočtu numerického řešení Barenblattovy úlohy na jednotlivých sítích při použití různých řešičů a schématu bez upwindové stabilizace. Časový krok je volen $\Delta t = 1{,}25\,\text{s}$ na síti $N_x = 400$ a pro ostatní sítě $\Delta t \sim h^2$.

4.1.3 Výsledky s upwindovou stabilizací

Při odvozování numerického schématu v předchozí části kapitoly 2 jsme předpokládali, že na studované oblasti Ω v libovolném čase platí $\rho > 0$. Při nastavování testovací úlohy 4.1.1 pro základní numerické schéma popsané v první části kapitoly 2 jsme proto volili počáteční čas tak, aby nosič řešení pokrýval celou oblast. Pokud veličina ρ degeneruje, je potřeba použité numerické schéma vhodně modifikovat. Motivací z praxe je např. modelování vícefázového proudění v případech, kdy se některá z fází v části systému nenachází.

Nejprve porovnáme základní numerické schéma s modifikovaným schématem popsaným v kapitole 2.6 pomocí identické úlohy jako v kapitole 4.1.1. Dle tab. 4.4 vidíme, že při použití řešiče s upwindovou stabilizací je konvergence schématu pouze prvního řádu.

Použití upwindové stabilizace dále umožňuje modelování úloh s degenerovanou hustotou ρ, konvergenci modifikovaného schématu pro tento případ ověříme opět pomocí Barenblattovy úlohy. Změnou počátečního a koncového času oproti úloze z kapitoly 4.1.1 docílíme toho, že nosič řešení bude po celou dobu simulace uvnitř sledované oblasti a numerické řešení nebude ovlivněno okrajovými podmínkami. Zvolíme počáteční čas $t_0 = 10\,\text{s}$, koncový čas $T_f = 3 \cdot 10^3\,\text{s}$ a časový krok Δt volíme na jednotlivých sítích $4\times$ menší oproti původní úloze, tedy $\Delta t = 0{,}3125\,\text{s}$ pro síť $N_x = 400$ a pro ostatní sítě $\Delta t \sim h^2$. Zmenšením časového kroku dosáhneme vyšší přesnosti numerického řešení pro úlohu s rychlými změnami stavu systému během vývoje. Dle výsledků v tab. 4.5 je konvergence v normách L^1 a L^2 pro tuto úlohu také prvního řádu, v normě L^∞ se EOC postupně zvyšují se zjemňováním sítě.

N_x	h	$\|E_h\|_1$	EOC$_1$	$\|E_h\|_2$	EOC$_2$	$\|E_h\|_\infty$	EOC$_\infty$
25	4	$2{,}890 \cdot 10^{+00}$		$3{,}043 \cdot 10^{-02}$		$3{,}576 \cdot 10^{-04}$	
			0,663		0,656		0,617
50	2	$1{,}825 \cdot 10^{+00}$		$1{,}931 \cdot 10^{-02}$		$2{,}332 \cdot 10^{-04}$	
			0,848		0,846		0,832
100	1	$1{,}014 \cdot 10^{+00}$		$1{,}074 \cdot 10^{-02}$		$1{,}310 \cdot 10^{-04}$	
			0,927		0,926		0,921
200	0,5	$5{,}333 \cdot 10^{-01}$		$5{,}654 \cdot 10^{-03}$		$6{,}919 \cdot 10^{-05}$	
			0,965		0,965		0,963
400	0,25	$2{,}732 \cdot 10^{-01}$		$2{,}896 \cdot 10^{-03}$		$3{,}550 \cdot 10^{-05}$	

Tabulka 4.4: EOC v normách L^1, L^2 a L^∞ při použití schématu s upwindovou stabilizací pro úlohu s počátečním časem $t_0 = 3 \cdot 10^3\,\mathrm{s}$ a koncovým časem $T_f = 15 \cdot 10^3\,\mathrm{s}$. Časový krok je volen $\Delta t = 1{,}25\,\mathrm{s}$ na nejjemnější síti $N_x = 400$ a pro hrubší sítě $\Delta t \sim h^2$.

N_x	h	$\|E_h\|_1$	EOC$_1$	$\|E_h\|_2$	EOC$_2$	$\|E_h\|_\infty$	EOC$_\infty$
25	4	$1{,}674 \cdot 10^{+01}$		$1{,}772 \cdot 10^{-01}$		$2{,}147 \cdot 10^{-03}$	
			1,147		1,014		0,162
50	2	$7{,}562 \cdot 10^{+00}$		$8{,}770 \cdot 10^{-02}$		$1{,}919 \cdot 10^{-03}$	
			0,315		0,372		0,359
100	1	$6{,}081 \cdot 10^{+00}$		$6{,}779 \cdot 10^{-02}$		$1{,}496 \cdot 10^{-03}$	
			1,101		0,893		0,414
200	0,5	$2{,}835 \cdot 10^{+00}$		$3{,}651 \cdot 10^{-02}$		$1{,}123 \cdot 10^{-03}$	
			1,041		0,966		0,702
400	0,25	$1{,}378 \cdot 10^{+00}$		$1{,}869 \cdot 10^{-02}$		$6{,}900 \cdot 10^{-04}$	

Tabulka 4.5: EOC v normách L^1, L^2 a L^∞ při použití schématu s upwindovou stabilizací pro úlohu s počátečním časem $t_0 = 10\,\mathrm{s}$ a koncovým časem $T_f = 3 \cdot 10^3\,\mathrm{s}$. Časový krok je volen $\Delta t = 0{,}3125\,\mathrm{s}$ na nejjemnější síti $N_x = 400$ a pro hrubší sítě $\Delta t \sim h^2$.

4.2 Stlačitelné proudění bez gravitace

V předchozí kapitole jsme se zabývali úlohou ve speciálním tvaru, aby bylo možné využít znalosti analytického řešení. Dále se budeme zabývat modelovými úlohami s hodnotami koeficientů, které se více blíží hodnotám ze skutečného světa.

V úloze (1.7) se zabýváme prouděním stlačitelné tekutiny porézním prostředím, přičemž předepisujeme stavovou rovnici ideálního plynu (1.5). Při modelování proudění vzduchu pomocí výše popsaného modelu volíme molární hmotnost tekutiny $M = 28{,}96\,\mathrm{g \cdot mol^{-1}}$, pro simulace popsané v této kapitole volíme teplotu systému $T = 300\,\mathrm{K}$ a pro úplnost zde uvádíme také hodnotu molární plynové konstanty, která je přibližně rovna $R = 8{,}3144621\,\mathrm{J \cdot mol^{-1} \cdot K^{-1}}$. Při dané teplotě T je dynamická viskozita vzduchu přibližně rovna $\mu = 18{,}7 \cdot 10^{-6}\,\mathrm{Pa \cdot s}$.

Úlohu řešíme na obdélníkové oblasti Ω o rozměrech $100 \times 100\,\mathrm{m}^2$ bez vlivu gravitace ($\boldsymbol{g} = 0$), což v praxi odpovídá zkoumání proudění v horizontální vrstvě. Pro jednoduchost předpokládáme homogenní izotropní prostředí s porozitou $\Phi = 0{,}4$ a permeabilitou $k = 10^{-10}\,\mathrm{m}^2$.

Počáteční stav systému předepisujeme pomocí konstantní funkce $p_0 = 10^5\,\mathrm{Pa}$ na Ω. Levou, dolní a pravou stranu obdélníkové oblasti Ω předpokládáme nepropustné, požadujeme tedy splnění podmínky $q_{Neu} = 0$. Na horní straně oblasti předepisujeme Dirichletovu podmínku jako lineární přechod mezi hodnotami $p_1 = 10^5\,\mathrm{Pa}$ v levém horním a $p_2 = 1{,}01 \cdot 10^5\,\mathrm{Pa}$ v pravém horním rohu oblasti Ω, čímž simulujeme vítr foukající podél zkoumané oblasti. Pomocí numerické simulace zkoumáme vývoj systému za těchto podmínek v časovém intervalu $[0, T_f]$ pro koncový čas $T_f = 10^4\,\mathrm{s}$.

4.2.1 Numerické řešení

Výše popsanou úlohu řešíme pomocí `TNL` řešiče nejprve na síti o 100×100 elementech s časovým krokem $\Delta t = 30\,\mathrm{s}$. Po zkušenosti s Barenblattovou úlohou 4.1 volíme následující parametry GMRES řešiče: zastavovací parametr $\varepsilon = 10^{-13}$, restartování po $m = 10$ iteracích a minimální počet iterací $m_0 = 5$. S těmito parametry dosahujeme dobrých výsledků i na jemnějších sítích při zmenšování časového kroku $\Delta t \sim h^2$.

Časový vývoj řešení pro síť $N_x = 100$ je zobrazen na obr. 4.3. Zpočátku se rozložení tlaku v oblasti mění relativně rychle a po vyrovnání rozdílu mezi počáteční podmínkou a Dirichletovou podmínkou na horní straně oblasti se rychlost změn postupně zmenšuje. Řešení se postupně blíží stacionárnímu stavu, kterého se prakticky dosáhne v čase $t = 10^4\,\mathrm{s}$.

4.2.2 Srovnání výkonu

Pro úlohu bez gravitace dále podrobněji zkoumáme efektivitu `TNL` řešiče z hlediska výpočetních časů při použití různého hardware. Simulaci popsanou výše spustíme s identickými parametry postupně na několika různých počítačích, jejichž hlavní parametry jsou shrnuty v tab. 4.6. Naším hlavním cílem je prokázat urychlení výpočtu při použití GPU namísto CPU a případně charakterizovat hardware, na kterém k urychlení nedojde.

Označení		Hardware	Počet jader	Frekvence
A	$\mathrm{CPU_A}$	Intel Core i7-3770K	4†	3.5 GHz
	$\mathrm{GPU_{A1}}$	NVIDIA Tesla K40c	2880	745 MHz
	$\mathrm{GPU_{A2}}$	NVIDIA Tesla K20c	2496	706 MHz
B	$\mathrm{CPU_B}$	Intel Core 2 Duo E6600	2†	2.4 GHz
	$\mathrm{GPU_B}$	NVIDIA GeForce GTX 590	2×512‡	1215 MHz
C	$\mathrm{CPU_C}$	Intel Core i5-2430M	2†	2.4 GHz
	$\mathrm{GPU_C}$	NVIDIA GeForce GT 525M	96	950 MHz
D	$\mathrm{CPU_D}$	AMD Phenom II X6 1100T	6†	3.3 GHz
	$\mathrm{GPU_D}$	NVIDIA Tesla C2070	448	1147 MHz

† Výpočty na CPU vždy využívají pouze 1 jádro.

‡ Karta GeForce GTX 590 obsahuje celkem 1024 jader rozdělených do dvou samostatných GPU. Knihovna `TNL` v současné verzi dokáže využít pouze 1 GPU, tedy 512 jader.

Tabulka 4.6: Hardware použitý k výpočtům.

Absolutní doby výpočtu pro danou simulaci na sítích s $N_x \times N_x$ elementy, $N_x = 100$, $N_x = 200$, $N_x = 400$ a $N_x = 800$ jsou uvedeny v tab. 4.7. V jednotlivých sloupcích jsou nejprve uvedeny časy pro CPU a poté pro GPU s označeními zavedenými v tab. 4.6. Relativní srovnání dob výpočtu pro úlohu bez gravitace pro jednotlivá CPU a GPU z tab. 4.6 uvádíme v tab. 4.8. Hodnoty uvedené v tab. 4.8 počítáme jako maximální doba výpočtu na dané síti dělená dobou výpočtu pro danou PU[3] a danou síť, tedy $r_{i,j} = \max_j\{a_{i,j}\}/a_{i,j}$, kde $a_{i,j}$ označuje absolutní dobu výpočtu na i-tém řádku a j-tém sloupci tab. 4.7.

[3]Processing Unit

Na malých sítích při použití GPU není znát urychlení výpočtu oproti CPU, s rostoucím počtem elementů sítě se rozdíl zvětšuje a na nejjemnější síti $N_x = 800$ je výpočet na GPU (pomineme-li GPU_C) řádově $10\times$ rychlejší. Dále pokud blíže srovnáme časy dosažené na počítačích s označením A a B, pozorujeme přibližně trojnásobný rozdíl mezi nejlepším a nejhorším časem na CPU (CPU_A vs. CPU_B), ale rozdíl mezi odpovídajícími časy na GPU_A1 a GPU_B je pouze 30%, což přibližně odpovídá teoretickému rozdílu ve výkonnosti těchto GPU. Doba výpočtu na GPU tedy není příliš ovlivněna výkonem CPU na daném počítači a celkové srovnání mezi všemi CPU i GPU v tab. 4.8 má proto dobrý smysl.

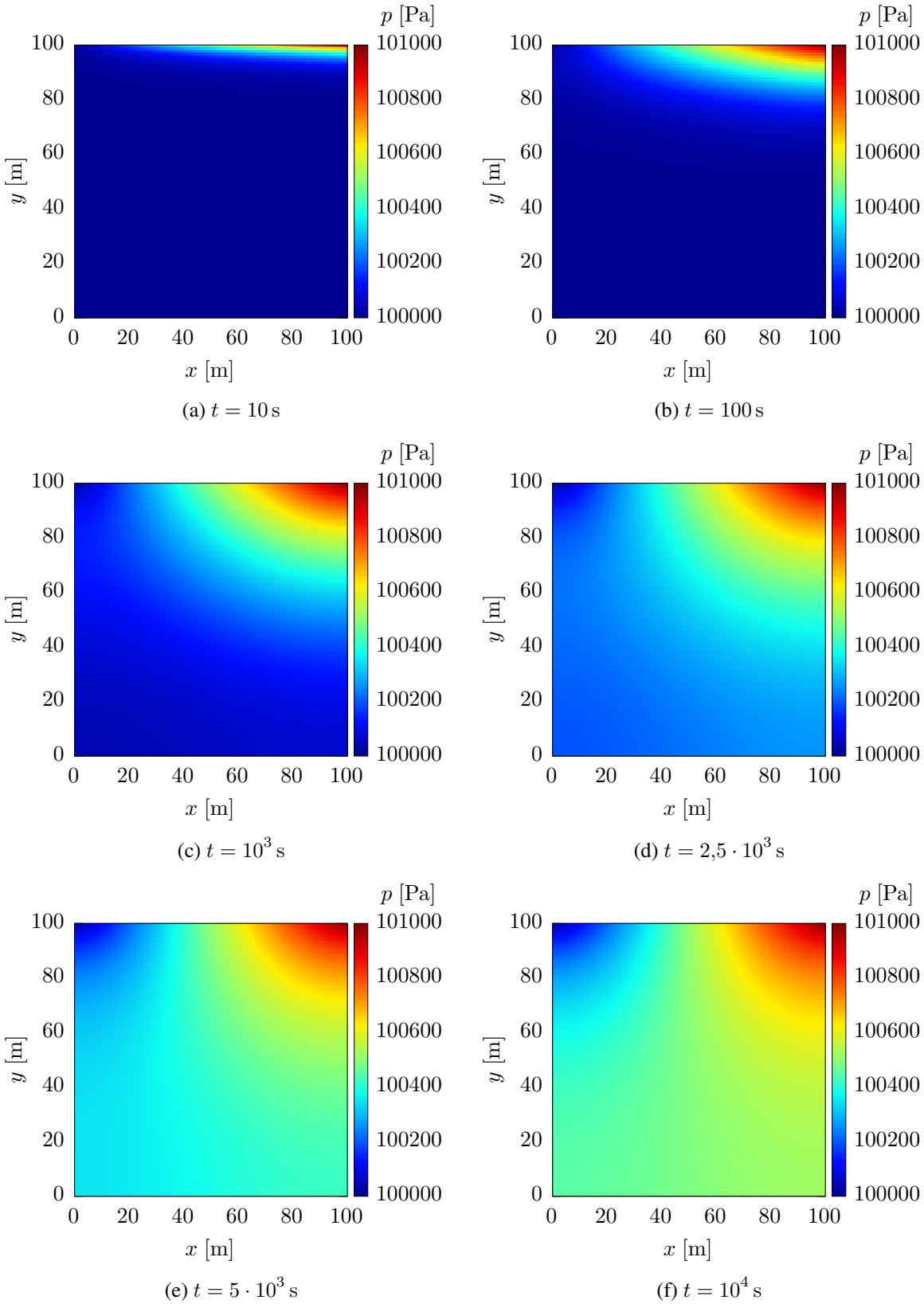
Mezi procesory se ukázal být nejrychlejším procesor s označením CPU_A, naopak na CPU_D bylo pro danou úlohu dosaženo výrazně horších výsledků, což je způsobeno zejména malou pamětí cache. Na počítači s označením A, kde bylo dosaženo nejlepších časů na CPU i GPU, je pro síť $N_x = 800$ paralelní výpočet na GPU_A1 přibližně $11\times$ rychlejší oproti CPU. Rozdíl 15% mezi GPU_A1 a GPU_A2 odpovídá rozdílu v počtu jader těchto GPU. Na GPU starší generace (GPU_B a GPU_D) bylo dosaženo přibližně o 30% horších časů ve srovnání s nejlepším časem dosaženým na GPU_A1. Výkon GPU_C je ovlivněn zejména malým počtem jader a pomalou pamětí.

N_x	CPU_A	CPU_B	CPU_C	CPU_D	GPU_A1	GPU_A2	GPU_B	GPU_C	GPU_D
100	8,0	17,6	11,9	13,0	5,4	5,7	6,9	21,7	5,2
200	80,9	204,7	122,7	182,8	18,3	19,8	25,7	128,0	23,1
400	1 055,9	3 193,2	1 705,0	2 405,8	135,0	154,3	195,3	1 331,9	191,2
800	17 834,0	52 851,2	25 074,7	37 574,5	1 557,7	1 821,1	2 335,2	18 565,8	2 380,0

Tabulka 4.7: Absolutní doby výpočtu v sekundách pro úlohu bez gravitace na jednotlivých PU z tab. 4.6.

N_x	CPU_A	CPU_B	CPU_C	CPU_D	GPU_A1	GPU_A2	GPU_B	GPU_C	GPU_D
100	2,7	1,2	1,8	1,7	4,0	3,8	3,2	1,0	4,2
200	2,5	1,0	1,7	1,1	11,2	10,4	8,0	1,6	8,8
400	3,0	1,0	1,9	1,3	23,7	20,7	16,4	2,4	16,7
800	3,0	1,0	2,1	1,4	33,9	29,0	22,6	2,9	22,2

Tabulka 4.8: Relativní srovnání dob výpočtu pro úlohu bez gravitace na jednotlivých PU z tab. 4.6. Hodnoty počítáme jako maximální doba výpočtu na dané síti dělená dobou výpočtu pro danou PU a danou síť, tedy $r_{i,j} = \max_j\{a_{i,j}\}/a_{i,j}$, kde $a_{i,j}$ označuje absolutní dobu výpočtu na i-tém řádku a j-tém sloupci tab. 4.7.



(a) $t = 10\,\mathrm{s}$

(b) $t = 100\,\mathrm{s}$

(c) $t = 10^3\,\mathrm{s}$

(d) $t = 2{,}5 \cdot 10^3\,\mathrm{s}$

(e) $t = 5 \cdot 10^3\,\mathrm{s}$

(f) $t = 10^4\,\mathrm{s}$

Obrázek 4.3: Vývoj tlaku vzduchu p v čase pro úlohu bez gravitace 4.2.

4.3 Stlačitelné proudění s gravitací

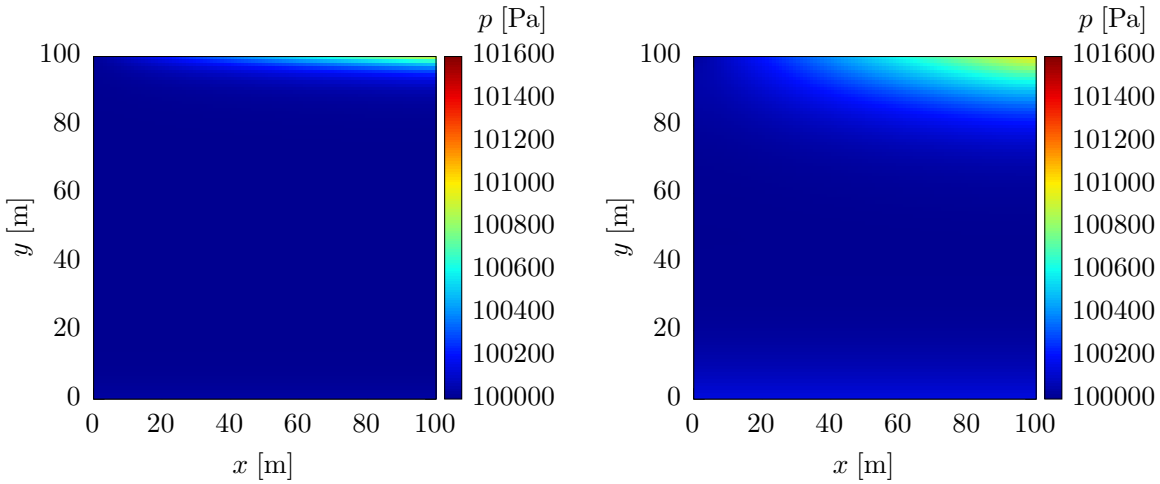
V poslední úloze se zabýváme modelováním proudění stlačitelné tekutiny porézním prostředím při uvažování gravitačního vlivu. Konkrétní parametry simulace zvolíme stejné jako v kapitole 4.2, včetně všech parametrů `TNL` řešiče, a změníme pouze složky vektoru gravitačního zrychlení $\boldsymbol{g}$. Při výpočtu prvků lokálního vektoru $\boldsymbol{G}_K$ pro obdélníkový element v kapitole 2.2.5 jsme předpokládali vektor gravitačního zrychlení ve tvaru $\boldsymbol{g} = (0, -g_y)$, zkoumáme tedy proudění ve vertikální vrstvě. Pro hodnotu $g_y = 9{,}81\,\mathrm{m}\cdot\mathrm{s}^{-2}$ je na obr. 4.4 zobrazeno numerické řešení této úlohy na síti o 100×100 elementech. Systém opět spěje ke stacionárnímu stavu, kterého se prakticky dosáhne v čase $t = 10^4\,\mathrm{s}$. Vlivem gravitace tlak tekutiny v systému klesá s rostoucí výškou nad nepropustnou spodní stranou oblasti, v horní části oblasti je rozložení tlaku ovlivněno předepsanou okrajovou podmínkou na horní straně oblasti.

Vliv gravitace představuje v rovnici (1.7) advekční člen, který má vliv na stabilitu numerického schématu. Dále tedy sledujeme chování řešičů v závislosti na síle gravitačního vlivu. Na síti o 400×400 elementech jsme řešili úlohu popsanou v předchozím odstavci a volili jsme různé hodnoty parametru g_y. Při použití `TNL` řešiče je dle tab. 4.9 patrná závislost doby výpočtu na parametru g_y, pro $g_y = 5000\,\mathrm{m}\cdot\mathrm{s}^{-2}$ navíc řešení lineárních soustav v jednotlivých časových krocích zabírá čím dál tím víc iterací (v prvním časovém kroku 30 iterací, v posledním 3500). Při použití `UMFPACK` řešiče je sice pro malé hodnoty g_y doba výpočtu nezávislá na g_y, ale pro $g_y > 10000\,\mathrm{m}\cdot\mathrm{s}^{-2}$ po několika časových krocích v důsledku numerických chyb selže řešič soustav lineárních rovnic. Základní schéma bez upwindové stabilizace popsané v kapitole 2 tedy není stabilní pro silně advekční úlohy.

Při použití upwindové stabilizace se schéma stabilizuje i pro extrémní hodnoty g_y, ale doba výpočtu je náchylná na volbu časového kroku vzhledem k počáteční podmínce. Pro „nepřesnou“ počáteční podmínku je potřeba zvolit velmi krátký časový krok, aby vůbec řešič lineárních soustav v prvním časovém kroku konvergoval, přitom pro následující časové kroky je takto zvolený časový krok zbytečně malý a lineární řešič konverguje velmi rychle. Do budoucna by proto bylo vhodné implementovat adaptivní volbu časového kroku pro advekční úlohy.

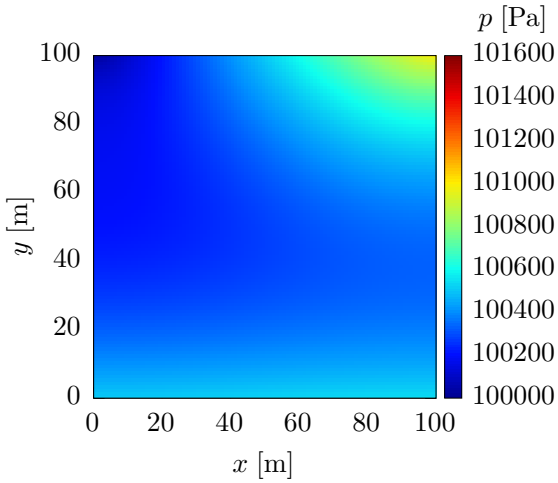
$g_y[\mathrm{m}\cdot\mathrm{s}^{-2}]$	Doba výpočtu [s]
0	226,3
10	270,3
100	524,8
500	1 037,0
1000	1 682,3
5000	38 480,6

Tabulka 4.9: Doby výpočtu při použití `TNL` řešiče bez upwindové stabilizace v závislosti na síle gravitace.

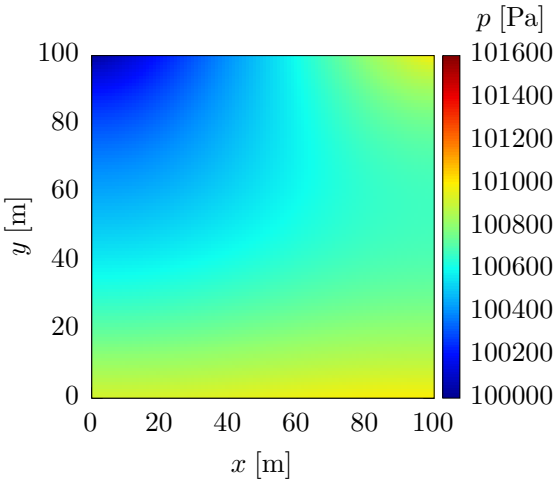


(a) $t = 10\,\mathrm{s}$

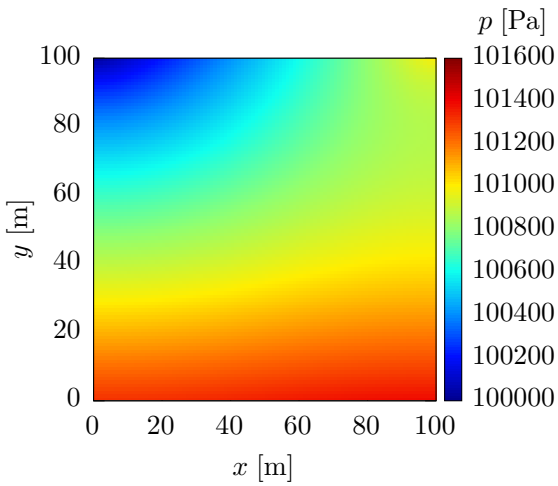
(b) $t = 100\,\mathrm{s}$



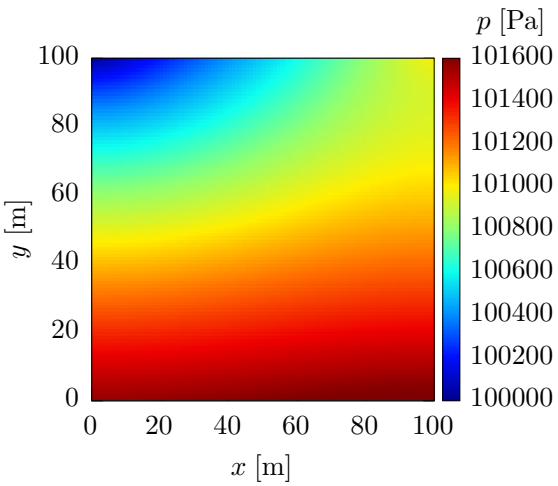
(c) $t = 10^3\,\mathrm{s}$



(d) $t = 2{,}5 \cdot 10^3\,\mathrm{s}$



(e) $t = 5 \cdot 10^3\,\mathrm{s}$



(f) $t = 10^4\,\mathrm{s}$

Obrázek 4.4: Vývoj tlaku vzduchu p v čase pro úlohu s gravitací 4.3.

Závěr

V práci jsme se zabývali modelováním stlačitelného jednofázového proudění v porézním prostředí a implementací paralelního řešiče na GPU. V první kapitole věnující se formulaci zkoumaného problému jsme na základě rovnice kontinuity, Darcyho zákona a stavové rovnice sestavili soustavu parciálních diferenciálních rovnic popisující chování zkoumaného systému. Tuto soustavu jsme dále doplnili o počáteční a okrajové podmínky.

Pro řešení zformulované úlohy jsme ve druhé kapitole sestavili semi-implicitní numerické schéma založené na kombinaci smíšené hybridní metody konečných prvků a metody konečných objemů pro prostorovou diskretizaci. Pro časovou diskretizaci jsme použili zpětnou Eulerovu metodu a získané diskrétní schéma jsme linearizovali v čase pomocí metody zamrzlých koeficientů. Vzniklou soustavu lineárních rovnic jsme dále upravili pomocí metody hybridizace a snížili tak počet rovnic v soustavě. V poslední části této kapitoly jsme popsali modifikaci základního numerického schématu pomocí upwindové stabilizace.

Třetí kapitola této práce se věnuje implementaci numerického schématu bez upwindové stabilizace. V první části kapitoly je popsána implementace numerického schématu pomocí vlastního kódu v jazyce C++ s využitím knihovny `UMFPACK` pro řešení řídkých soustav lineárních rovnic. Tento řešič sloužil především jako testovací nástroj a odrazový můstek pro netriviální implementaci paralelního řešiče pomocí knihovny `TNL`, která je popsána ve druhé části kapitoly. Knihovna `TNL` poskytuje mimo jiné efektivní iterační řešiče pro řešení řídkých soustav lineárních rovnic a jednotné rozhraní pro architekturu `CUDA`, takže výsledný program lze spustit jak na CPU, tak i masivně paralelně na GPU. Implementace a optimalizace paralelního řešiče zabrala podstatnou část vývojového času, především kvůli specifické architektuře GPU z hlediska práce s pamětí a netriviální paralelizaci numerického schématu.

Konvergenci obou variant numerického schématu a správnost následné implementace jsme ověřili pomocí experimentální analýzy konvergence pro Barenblattovu úlohu se známým analytickým řešením. Konvergence numerického schématu bez upwindové stabilizace je druhého řádu, při použití upwindové stabilizace je konvergence pouze prvního řádu, což odpovídá výsledkům v literatuře. Z hlediska efektivity je pro danou úlohu na síti se 400×400 elementy (tj. 320800 stupňů volnosti) použití `TNL` řešiče na CPU přibližně $7\times$ rychlejší oproti `UMFPACK` řešiči, dalšího přibližně sedminásobného urychlení výpočtu je možné dosáhnout spuštěním paralelního `TNL` řešiče na GPU. Celkově je tedy urychlení při použití `TNL`-GPU řešiče téměř $50\times$ oproti `UMFPACK` řešiči při srovnatelné chybě numerického řešení. Pro úlohu se skutečnými materiálovými parametry na síti s 800×800 elementy (tj. 1281600 stupňů volnosti) je urychlení `TNL`-GPU řešiče oproti `TNL`-CPU $10\times$ až $30\times$ v závislosti na použitém hardware. Na advekční úloze s gravitačním vlivem byla prokázána nestabilita základní varianty numerického schématu bez upwindové stabilizace. Při použití upwindové stabilizace je schéma stabilní, ale pro praktické využití je potřeba implementovat adaptivní volbu časového kroku.

Do budoucna bude cílem využít získané poznatky paralelizace MHFEM/FVM schématu v rámci knihovny `TNL` pro implementaci obecného modelu vícefázového kompozičního proudění v porézním prostředí.

Literatura

[1] Ľubomíra Balková. *Lineární algebra 2*. (In collaboration with students J. Krásenský and J. Klinkovský). Nakladatelství ČVUT, Praha, 2014. ISBN: 978-80-01-05441-3.

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

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

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

[5] Timothy A. Davis. 'A Column Pre-ordering Strategy for the Unsymmetric-pattern Multifrontal Method'. In: *ACM Trans. Math. Softw.* 30.2 (June 2004), pages 165–195. ISSN: 0098-3500. DOI: 10.1145/992200.992205. URL: http://doi.acm.org/10.1145/992200.992205.

[6] Timothy A. Davis. 'Algorithm 832: UMFPACK V4.3—an Unsymmetric-pattern Multifrontal Method'. In: *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.

[7] Timothy A. Davis. 'Umfpack version 4.4 user guide'. In: *Department of Computer and Information Science and Engineering, University of Florida, Gainesville, FL* (2005).

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

[9] Zhangxin Chen, Guanren Huan and Yuanle Ma. *Computational Methods for Multiphase Flows in Porous Media*. Volume 2. SIAM, 2006.

[10] Randall J. LeVeque. *Finite volume methods for hyperbolic problems*. Volume 31. Cambridge university press, 2002.

[11] Tomáš Oberhuber and Martin Heller. 'Improved row-grouped CSR format for storing of sparse matrices on GPU'. In: *Proceedings of Algoritmy*. 2012, pages 282–290. ISBN: 978-80-227-3742-5.

[12] Tomáš Oberhuber, Atsushi Suzuki, Jan Vacata et al. 'Image segmentation using CUDA implementations of the Runge-Kutta-Merson and GMRES methods'. In: *Journal of Math-for-Industry* (2011), pages 73–79.

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

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

[15] Ole Tange. 'GNU Parallel - The Command-Line Power Tool'. In: *The USENIX Magazine* 36.1 (Feb. 2011), pages 42–47. DOI: 10.5281/zenodo.16303. URL: http://www.gnu.org/s/parallel.

[16] Thomas Williams, Colin Kelley and many others. *Gnuplot 5.0: an interactive plotting program*. Jan. 2015. URL: http://gnuplot.sourceforge.net/.

[17] Anis Younes, Philippe Ackerer and François Lehmann. 'A new mass lumping scheme for the mixed hybrid finite element method'. In: *International journal for numerical methods in engineering* 67.1 (2006), pages 89–107.

[18] Qiang Zhang and Zi-Long Wu. 'Numerical simulation for porous medium equation by local discontinuous Galerkin finite element method'. In: *Journal of Scientific Computing* 38.2 (2009), pages 127–148.