TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Návrh a implementace propojení softwarů realizujících modely transportu a reakcí

Diplomová práce

Studijní program: N2612 – Elektrotechnika a informatika
Studijní obor: 1802T007 – Informační technologie

Autor práce: **Bc. Michal Štrajt**
Vedoucí práce: doc. Ing. Jan Šembera, Ph.D.

Liberec 2016

TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky a mezioborových studií
Akademický rok: **2015/2016**

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Michal Štrajt**
Osobní číslo:	**M14000180**
Studijní program:	**N2612 Elektrotechnika a informatika**
Studijní obor:	**Informační technologie**
Název tématu:	**Návrh a implementace propojení softwarů realizujících modely transportu a reakcí**
Zadávající katedra:	**Ústav mechatroniky a technické informatiky**

Zásady pro vypracování:

1. Na základě poznatků získaných při řešení semestrální práce navrhněte propojení softwarů Transport a React s využitím databáze výsledků reakčního modelování. Interface bude obsahovat algoritmus k rozhodování, zda požadovaný výpočet provede software React, anebo bude použit některý výsledek z již dříve provedených výpočtů uložený v databázi.
2. Návrh implementujte.
3. Propojení otestujte na vhodně vybrané úloze.

Rozsah grafických prací: **dle potřeby dokumentace**

Rozsah pracovní zprávy: **cca 40–50 stran**

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

[1] **Bethke, C.M. and S. Yeakel, 2009, The Geochemist's Workbench®, Version 8.0: Reactive Transport Modeling Guide. Hydrogeology Program, University of Illinois, Urbana, 106 p.**

[2] **Vratislav Žabka, Program Transport v2.2, dokumentace, TU v Liberci, Liberec 2011.**

[3] **Etapová zpráva projektu č. TA02021132 "Mobilita kontaminantů a dalších složek prostředí – integrace do expertního systému využívajícího transportně-reakční modelování" za rok 2014, TUL 2015.**

Vedoucí diplomové práce: **doc. Ing. Jan Šembera, Ph.D.**
Ústav mechatroniky a technické informatiky

Datum zadání diplomové práce: **10. října 2015**

Termín odevzdání diplomové práce: **16. května 2016**

prof. Ing. Václav Kopecký, CSc.
děkan

L.S.

Technická univerzita v Liberci - 14 -

doc. Ing. Milan Kolář, CSc.
vedoucí ústavu

V Liberci dne 10. října 2015

Prohlášení

Byl jsem seznámen s tím, že na mou diplomovou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

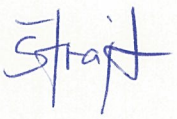
Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé diplomové práce pro vnitřní potřebu TUL.

Užiji-li diplomovou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti TUL; v tomto případě má TUL právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Diplomovou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím mé diplomové práce a konzultantem.

Současně čestně prohlašuji, že tištěná verze práce se shoduje s elektronickou verzí, vloženou do IS STAG.

Datum: 12.5.2016

Podpis: 

Poděkování

Na tomto místě bych chtěl poděkovat doc. Ing. Janu Šemberovi, Ph.D. za vedeni práce a za užitečné rady a připomínky.

Abstrakt

Program Transport je nástroj určený k modelování kolonových experimentů. Při řešení kolonového experimentu je v každém časovém okamžiku experimentu řešena transportní a reakční část výpočtu. K výpočtu reakční části je využíváno volání externího programu React. Volání externího programu je ovšem časově náročné, proto je třeba celý proces optimalizovat.

V práci je navržen a implementován algoritmus, který rozšiřuje propojení programů Transport a React o komunikaci s databází. Do databáze jsou ukládány výsledky reakčních výpočtů a následně čteny namísto opakovaného volání programu React. Tím dochází ke snížení časové náročnosti programu. V práci je také rozebráno, za jakých podmínek je algoritmus efektivní.

Klíčová slova: program Transport, optimalizace, kolonový experiment

Abstract

The Transport software is designed to calculate models of convoy experiments. In every time step of the calculation, the transport and the reaction part of the calculation are solved. An external programme called React is used to calculate the reaction part of the calculation. However, calling an external program takes long time. That is why optimizing the whole process is needed.

In this paper, an algorithm extending the link between the Transport and React softwares by database connection is designed and implemented. The results of the reaction part of the calculation are saved into the database and read from there. That prevents calling the React software repeatedly. Thereby the calculation duration is reduced. This paper contains the effectivity conditions analysis of this algorithm.

Key words: software Transport, optimizing, column experiment

Obsah

Seznam ilustrací

Seznam tabulek

Seznam zdrojových kódů

Úvod

Transportně reakční model kolonového experimentu může poskytnout informace o chování podzemních vod. Kolona, tedy objekt, který je kolonovým experimentem modelován, je objekt, v případě této práce válec, který je vyplněn horninovým prostředím a roztokem, který je v rovnováze s daným prostředím. Do kolony je z jedné strany (podstavy válce) vtláčena voda nebo roztok a z druhé strany stejné množství vody vytéká. Ostatní stěny kolony jsou nepropustné, transport je tedy prováděn jednorozměrně z jedné strany kolony na druhou.

K modelování těchto jevů byl již dříve vytvořen program Transport. Program Transport je od svého vzniku stále rozšiřován a jsou do něj přidávány stále nové funkcionality. Program Transport řeší transportní část kolonového modelu a k výpočtu reakční části volá externí program React z balíčku The Geochemist's Workbench.

K tomu, aby kolona mohla být programem simulována, byla diskretizována do menších objektů – buněk a výpočet je prováděn v časových krocích. V každém časovém kroku se provede nejprve transportní a poté reakční část výpočtu.

Reakční část výpočtu, vzhledem k tomu, že je realizována externím programem React, který je volán programem Transport, může být pro velký počet buněk kolony časově velmi náročná. Pro reálné použití programu je tedy nutné provést optimalizaci, která povede ke snížení časové náročnosti. Jeden typ časové optimalizace byl již v programu implementován dříve. Práce se zabývá druhým typem časové optimalizace.

Tento nový typ časové optimalizace využívá ukládání výsledků reakční části do databáze během výpočtu. Následně při dalším výpočtu reakční části mohou být výsledky přečteny z databáze, případně pomocí záznamu z databáze aproximovány. Tím dojde k tomu, že není pro získání výsledků reakční části volán tak často program React a časová náročnost celkově klesá.

K tomu, aby tento algoritmus fungoval je potřeba vybrat vhodnou databázi a vhodný vyhledávací klíč. Předpokladem pro efektivitu tohoto řešení totiž je, že vyhledání v databázi bude rychlejší než volání externího programu a výpočet provedený tímto programem.

Cílem práce je tedy vytvořit algoritmus, který bude rozhodovat, které výsledky budou načteny z databáze a které budou získány pomocí programu React. Tento algoritmus bude implementován do programu Transport jako druhý způsob časové optimalizace. Oba způsoby

časové optimalizace jsou na sobě nezávislé a po dokončení práce bude program poskytovat možnost vybrat si, kterým způsobem má být úloha řešena. Je tedy také nutné program Transport upravit tak, aby tato volba byla možná.

Kromě vlastní implementace je také nutné otestovat efektivitu tohoto nového rozšíření. Vzhledem k tomu, že použití databáze a aproximace výsledků s sebou přináší další možnosti, je třeba otestovat, v jakých případech je algoritmus efektivní a v jakých případech efektivní není.

1 Teoretický rozbor

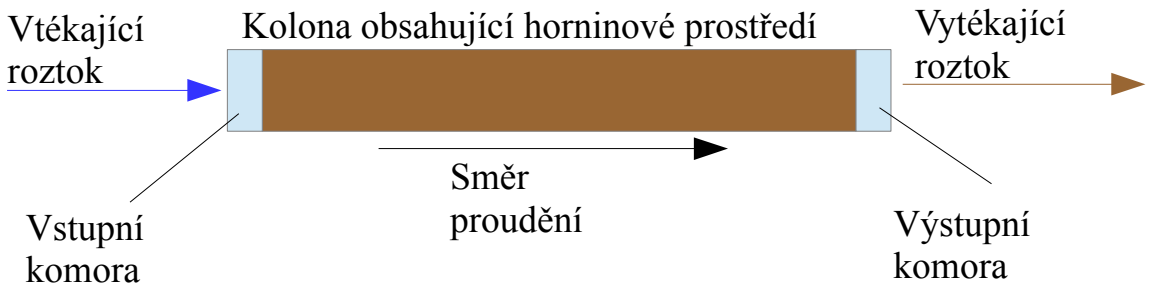
1.1 Program Transport

Práce se bude zabývat návrhem, implementací a testováním rozšíření programu Transport. Proto je programu věnována samostatná kapitola teoretického rozboru. Program Transport je vyvíjen na Technické univerzitě v Liberci přibližně od roku 2007. Program je vyvíjen v jazyce C# s použitím frameworku .NET verze 4.0. Od doby, kdy začal být program vyvíjen, prošel několika změnami. V následujících kapitolách je popsána nejnovější verze programu tak, jak vypadal před začátkem této práce. Popis úprav, které přináší tato práce, bude rozebrán až v kapitole 2, aby bylo možné rozlišit verze programu před zahájením práce a po jejím dokončení.

1.1.1 Kolonové experimenty

Program Transport je nástroj, který je určený k modelování transportně reakčních procesů, konkrétně experimentů probíhajících v koloně. Princip kolonového experimentu je následující. Celý experiment se odehrává v prostředí zvaném kolona. Tato kolona je, v případě programu Transport, válcového tvaru. Stěny válce jsou tvořeny nepropustným inertním materiálem. Uvnitř válce je horninové prostředí obsahující roztok, který je s tímto horninovým prostředím v rovnováze. Podstavy válce jsou propustné a slouží k tomu, aby skrz kolonu mohly proudit roztoky. Tato situace simuluje stav podzemní vody.

Do kolony vtéká přes vstupní komoru druhý roztok, obecně s jiným složením než roztok původní. Roztok protéká zadanou rychlostí skrz kolonu a mísí se s roztokem, který je již v koloně obsažen. Přes výstupní komoru pak stejné množství roztoku vytéká. Transport roztoků skrz kolonu je jednorozměrný. Jde tedy pouze o transport z jedné strany kolony na druhou. Situace je schematicky znázorněna na obrázku 1.



Obrázek 1: Schematický nákres kolonového experimentu

Pro výpočet takového modelu je v programu Transport zvolena metoda konečných objemů.

To znamená, že je kolona rozdělena do několika objektů – buněk. Buňky jsou řazeny za sebe ve směru proudění. Kolona je rozdělena rovnoměrně, všechny buňky mají tedy stejný objem.

Ke změnám v buňkách dochází v pravidelných časových intervalech – krocích. Po uplynutí každého časového kroku provede program Transport transportní část procesu, tedy posun roztoků o úsek, závislý na rychlosti proudění, ve směru proudění. Dále program provede míchání roztoků v každé z buněk.

Po dokončení transportní části umožní program výpočet části reakční. V reakční části se počítají proběhlé chemické reakce, tedy srážení a rozpouštění hornin. Reakční část výpočtu je provedena pro každou buňku zvlášť. Reakční část není realizována přímo v programu Transport, k jejímu výpočtu je volán externí program React z balíčku The Geochemist's Workbench.

Podrobnější popis simulace kolonových experimentů v programu Transport je možné najít v práci[1], během které byl program Transport také vyvíjen a je tedy jednou z prací, na kterou tato diplomová práce navazuje.

1.1.2 Popis programu

Popis programu Transport, který je zde uveden, je stručným výtahem z dokumentace[2] k programu, kde je možné najít podrobnější informace.

Pomocí konfiguračního souboru je možné programu Transport nastavit parametry kolony, jako jsou její rozměry, počet buněk, do kterých je kolona rozdělena, parametry k počítání trojí porózity a další. Dále je možné nastavit, které všechny transportní výpočty se mají pomocí programu provádět. Program nabízí například výpočet simulace rovnovážné sorpce, retardace nebo difuze. Po startu programu je načten konfigurační soubor a data jsou zobrazena v grafickém rozhraní, kde je možné je ještě před spuštěním samotného výpočtu upravit.

Program Transport využívá ke všem chemickým výpočtům komerční program React z balíčku The Geochemist's Workbench. Před spuštěním výpočtu v programu Transport je třeba vytvořit model pro React, který bude obsahovat informace o složení kolony na začátku výpočtu. Dále je třeba vytvořit modely pro React, které budou obsahovat složení vtláčených roztoků, jeden soubor pro výstupní komoru a jeden pro složení pórů. Všechny tyto modely jsou programem React vypočteny před zahájením výpočtu v programu Transport. Program Transport si uloží výsledky těchto modelů.

Program nabízí možnost vtláčet do kolony střídavě dva různé roztoky. Každý roztok je popsán ve vlastním souboru pro React. Informace o tom, který roztok je ve kterém čase do kolony vtláčen a jakou rychlostí kolonou protéká, se nastavují v textovém souboru, který je programem Transport načten před zahájením výpočtu.

Další možnost, kterou program Transport nabízí je výpočet výstupní baňky. Výstupní baňka je nádoba, umístěná za kolonou, do které vtéká roztok, který vytéká z kolony ven. Výpočet baňky je analyzován v časových intervalech. To znamená, že se baňka určitou dobu plní a poté, co uplyne daný interval se vyleje, analyzuje a plní znova. Obsah baňky se tedy nemusí shodovat s obsahem v poslední buňce kolony.

Program nabízí celkem tři metody výpočtu rekční složky. Metody se liší v tom, za jakých podmínek bude volán program React a spuštěn výpočet reakční složky. Těmito metodami jsou „míchání“, „chemie vždy“ a „čelo kontaminace“.

Během výpočtu pomocí metody „míchání“ program po celou dobu výpočtu provádí pouze transportní část výpočtu, reakční složka se nepočítá vůbec. Po skončení běhu programu jsou spočítány rovnováhy ve výstupní baňce.

Opakem této metody je metoda „chemie vždy“, která spouští výpočet reakční složky v každém časovém kroku a pro každou buňku. Tento výpočet je časově velmi náročný a pro jeho efektivní použití je vhodné jej optimalizovat.

U metody „čelo kontaminace“ se reakční program volá v buňkách, ve kterých se potkávají dva roztoky různého složení. U této metody se předpokládá, že právě v těchto místech bude docházet k největšímu množství chemických reakcí. Tato metoda se snaží využít výhod obou výše popsaných metod.

1.1.3 Program React a jeho spouštění

Program Transport při svých výpočtech využívá volání externího programu React z balíčku The Geochemist's Workbench. Program React je součástí standardní licence balíčku. React je program, který modeluje rovnovážné stavy a geochemické procesy ve vodném roztoku. Program dále nabízí možnost přidání kinetických parametrů k systému. Podrobnější popis programu je možné nalézt v jeho dokumentaci[3], pro účely programu Transport by však tento popis měl stačit.

Program Transport využívá externího programu React k výpočtu reakční části transportně

reakčního procesu. Program React je programem Transport volán po dokončení transportní části procesu pro každý časový krok, a pro každou buňku, ve které jsou splněny všechny podmínky vybrané metody spouštění. Metody spouštění jsou popsány v kapitole 1.1.2.

Ve verzi, ze které tato diplomová práce vychází, byly implementovány dva způsoby spouštění programu React. Tato diplomová práce rozšiřuje možnosti spouštění programu React. Toto rozšíření bude v práci podrobněji popsáno dále. Tato kapitola je věnována stavu programu Transport tak, jak vypadal před začátkem práce.

Obě verze spouštění programu React předpokládají, že uživatel před startem výpočtu připravil skript pro program React s názvem „ScriptVSE.rea“. Skript je sestaven ze tří částí, čtení dat, definování specifických vlastností modelu a zápis dat.

Části čtení a zápis dat jsou určeny pro komunikaci s programem Transport, uživatel by do ní tedy neměl zasahovat. Naopak do části definování specifických vlastností modelu může být uživatel nucen zasáhnout v případě, že chce počítat jiný model, než který byl počítán naposled. Specifickými vlastnostmi je myšleno zejména povolení a zakázání minerálů, které se mohou v modelu vyskytovat nebo například údaj o teplotě systému.

Skript předpokládá, že před jeho spuštěním byly programem Transport připraveny soubory, které udávají celkové složení kolony. V souborech jsou uvedeny koncentrace všech složek, které jsou programem Transport podporovány. Pro velmi jednoduché modely tyto soubory obsahují většinu údajů nulových, protože se složky v modelu vůbec nevyskytují.

SkriptVSE postupně prochází soubory se složením kolony a pro každý řádek souboru spouští výpočet. Jeden řádek souboru odpovídá jedné buňce kolony. Výsledky výpočtu jsou průběžně ukládány a po doběhnutí skriptu jsou na výstupu opět soubory, které obsahují celkové složení kolony, ve stejném formátu, jako byly vstupní soubory.

Použití tohoto skriptu přineslo výhodu, že pro celou kolonu je spuštěn právě jeden skript, místo toho, aby byl spouštěn skript pro každou buňku kolony. Program Transport tím pádem neztrácí tolik času opakovaným spouštěním skriptu a klesá tím jeho časová náročnost.

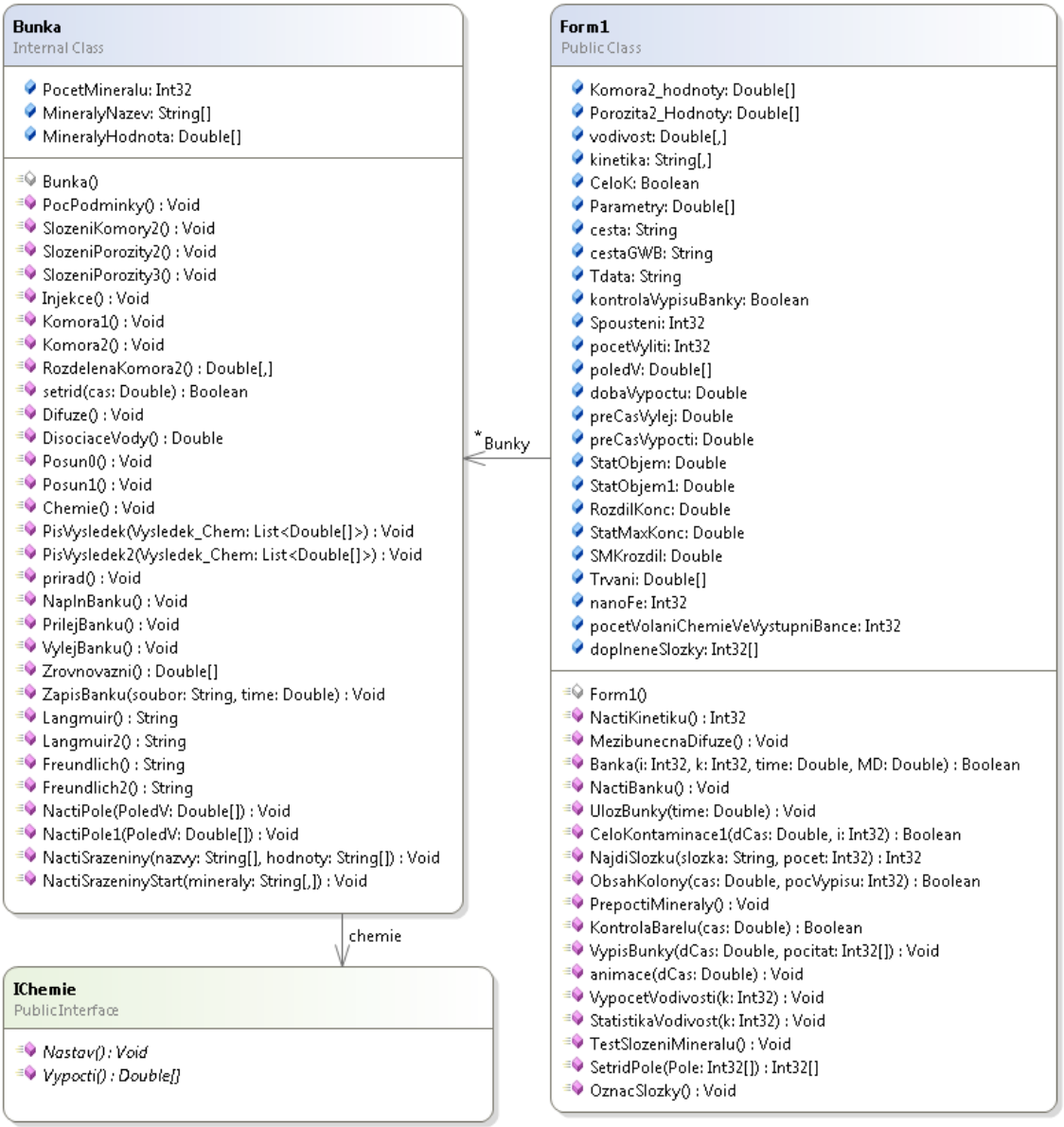
Jak bylo psáno výše, program Transport nabízí dva způsoby spouštění programu React. Liší se tím, zda je program React spouštěn s grafickým rozhraním nebo bez něj. Způsob volání programu React s grafickým rozhraním je výrazně pomalejší, může však, v případě, že se model

nevyvíjí tak, jak je očekáváno, poskytnout podrobnější informace o stavu kolony.

1.1.4 Struktura programu

V této kapitole je popsána struktura programu Transport z hlediska uspořádání zdrojových kódů a diagramu tříd. Zde je, podobně jako v kapitole 1.1.3 uveden stav programu tak, jak vypadal před zahájením této diplomové práce.

Kompletní diagram tříd je vzhledem k jeho rozsáhlosti uložen na přiloženém CD. Na obrázku 2 je zobrazen zjednodušený diagram tříd, který by měl stačit ke stručnému popisu. V diagramu na uvedeném obrázku jsou vyobrazeny pouze veřejné metody a atributy. Některým metodám byly z důvodů přehlednosti odebrány jejich parametry.



Obrázek 2: Zjednodušený diagram tříd programu Transport ve verzi 2.2

Hlavní třídou programu Transport je třída Form1. Tato třída obsahuje kompletní implementaci grafického rozhraní programu. Kromě toho třída obsahuje mnoho dalších atributů a metod, které s grafickým rozhraním přímo nesouvisí. Jsou to například atributy, do kterých jsou ukládána data určená k výpočtům programu, například atribut kinetika nebo cestaGWB. Dále jsou ve třídě metody, které provádějí vlastní výpočet, například metoda Transport, která je hlavní výpočetní metodou programu a metody pro práci se soubory, například NactiKinetiku.

Třída Bunka obsahuje informace o složení jedné buňky kolony a metody, které je možné nad buňkami provádět. Dále obsahuje instanci implementující rozhraní IChemie. Třídy, které implementují rozhraní IChemie zajišťují volání programu React v určený moment podle zvolené metody volání geochemického programu popsané v kapitole 1.1.2.

1.2 Databázové systémy

V práci bude k programu Transport implementováno rozšíření pro komunikaci s databází. Vzhledem k funkcionalitě, kterou má rozšíření nabízet, je vhodné uvažovat jak databáze relační, tak NoSQL databáze. V této kapitole budou obě skupiny stručně srovnány. Podrobné srovnání obou skupin databází je možné nalézt v článku[4], ze kterého byly informace čerpány.

První a nejdůležitější vlastností relační databáze je vlastnost ACID transakcí. Zkratka ACID pochází z počátečních písmen čtyř vlastností: atomicita (Atomicity), konzistence (Consistency), nezávislost (Isolation) a trvanlivost (Durability). Existují případy, kdy jsou ACID transakce nutností pro fungování aplikace využívající databázi. NoSQL databáze na rozdíl od databází relačních vlastnost ACID obecně neposkytují.

Díky atomicitě musí transakce proběhnout buď celá, nebo vůbec. Pokud byla databáze v konzistentním stavu, tak díky konzistenci musí být po provedení transakce také v konzistentním stavu. Nezávislost zajišťuje, že ostatní transakce uvidí provedené změny v databázi až po dokončení kterékoliv jiné transakce a nemůže přistupovat k částečným změnám. Díky trvanlivosti jsou změny v databázi uloženy natrvalo.

Další důležitou vlastností relačních databází je použití jazyka SQL, který je v mnoha databázích definován podobně a tím je zajištěn poměrně jednoduchý přechod od jedné relační databáze k druhé. NoSQL databáze žádný takový standardní jazyk nebo API neposkytují.

Relační databáze je možné popsat relačním modelem, kde jsou data uložena v tabulkách,

které mají řádky a sloupce. Naproti tomu NoSQL databáze jsou často uspořádány jako dvojice klíč – hodnota, který poskytuje rychlý přístup k datům, případě jako kolekce transparentních dokumentů a další. Dokumenty v rámci jedné kolekce mohou mít různou strukturu, tedy různý počet záznamů různé délky. Takový způsob uložení je výhodný tam, kde jsou data částečně strukturována nebo pro ukládání objektů z pohledu objektově orientovaného programování.

2 Praktická část

2.1 Úprava struktury programu

Původní struktura programu, jak byla popsána v kapitole 1.1.4, měla několik vlastností, které budou popsány v této kapitole. Mnoho těchto vlastností bylo důsledkem míchání grafického rozhraní a výpočetních funkcí v jedné třídě.

Během samotného výpočtu docházelo ke čtení dat přímo z komponent grafického rozhraní, jejich datová konverze a dosazení do výpočtu. Z toho vyplývá, že jakákoliv operace ze strany uživatele v době běhu výpočtu mohla teoreticky výpočet ovlivnit, případně dostat program do neočekávaného stavu. Dalším důsledkem ukládání dat tímto způsobem je fakt, že během výpočtu mohlo docházet k výjimkám způsobeným současným přístupem dvou vláken k jedněm datům.

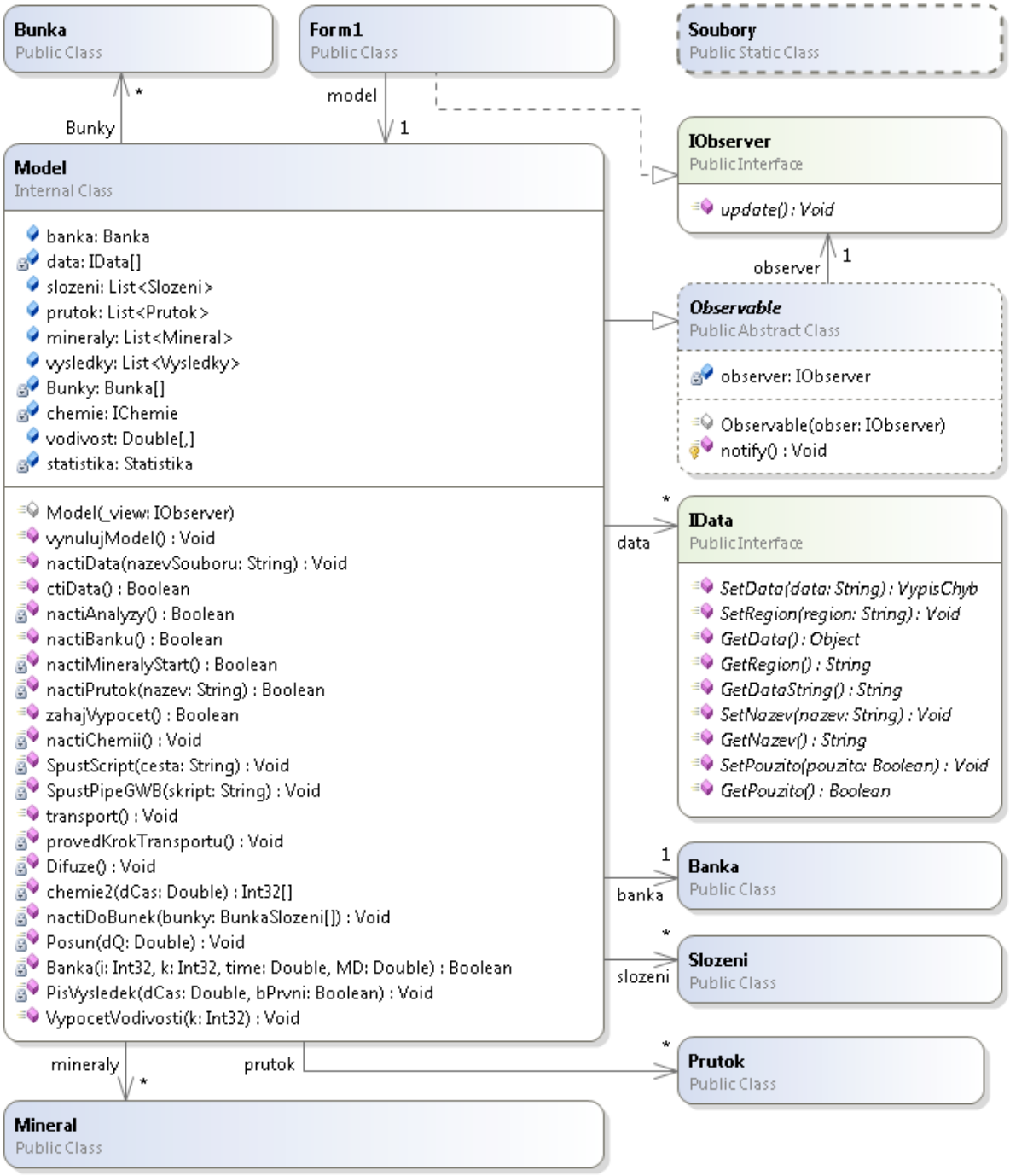
Vzhledem k tomu, že návrh programu byl, mimo popsané vlastnosti, také obtížně rozšiřitelný, bylo rozhodnuto, že před zanesením nové funkcionality bude struktura programu upravena především za účelem oddělení grafického rozhraní od výpočetních tříd. K tomu by měla posloužit implementace návrhového vzoru Observer.

Na obrázku 3 je uveden zjednodušený diagram tříd programu Transport po uvedených úpravách. Vzhledem k rozsáhlosti diagramu je v celé své podobě uložen na přiloženém CD a zde je uveden pouze zjednodušený diagram pro pochopení kontextu.

Třída Bunka zůstala, až na občasné změny datových typů, oproti verzi předchozí nezměněna. Třídě Form1 zůstaly pouze atributy a metody, které se týkají obsluhy grafického rozhraní. Z těchto důvodů jsou v diagramu uvedeny pouze hlavičky jmenovaných tříd.

Na rozdíl od původní verze programu, která data z konfiguračního souboru načítala přímo do grafického rozhraní, nová verze načítá tato data jako pole objektů typu IData do nově vzniklé třídy Model. Ke zkopírování údajů do grafického rozhraní pak slouží dvojice metod notify – update, které jsou součástí návrhového vzoru Observer.

Další novinkou ve struktuře programu je způsob práce se soubory. Práce se soubory je totiž velmi častou operací programu Transport a tím pádem existuje i velké množství metod k tomu určených. Z toho důvodu vznikla statická třída Soubory, ve které jsou všechny metody, které nějakým způsobem se soubory pracují. Tato třída vznikla zejména z důvodu přehlednosti kódu.



Obrázek 3: Zjednodušený diagram tříd programu Transport ve verzi 3.0

Do třídy Model byly extrahovány všechny výpočetní funkce a atributy k nim potřebné ze třídy Form1. Na diagramu na obrázku jsou z důvodu úspory místa vidět pouze ty nejdůležitější. Přesto je však názorně vidět práce programu od jeho spuštění, přes načtení konfiguračního souboru (metoda nactiData), načtení složení kolony (metoda ctiData), spuštění výpočtu (metoda transport), spouštění programu React (metoda SpustScript) až po uložení výsledků (metoda PisVysledek).

Ostatní, dosud nejmenované, třídy uvedené na diagramu slouží k uložení dat, které spolu souvisí. Pro ukázku je uvedena část kódu, jak byla tato data ukládána v předchozí verzi programu a jak jsou ukládána nyní.

Na první ukázce (viz Kód 1) je vidět část kódu ze třídy Form1. Je zde vidět soustava polí. Hodnoty do těchto polí jsou načítány z jednoho vstupního souboru a úzce spolu souvisí. Programátor v takovém případě musel pracovat s velkým množstvím objektů.

```
public class Form1
{
	private List<string> Tabulka_Nazvy;
	private double[, ,] Tabulka_Hodnoty; //mnozstvi1, mnozstvi2, slozky
	private double[] Tabulka_Alfy;
	private double[] Tabulka_Mw;
	private double[,] Tabulka_Sorpce;
	private List<double>[] Analyzy_Casy;
	private List<double>[] Analyzy_Hodnoty;
	…
}
```

Kód 1: Ukázka původního přístupu k datům složení

Stejná struktura je v novém kódu reprezentována seznamem objektů typu Slozeni (viz Kód 2). Takový kód umožní programátorovi přístup k totožným datům jednodušším způsobem.

```
public class Slozeni
{
	public string nazev { get; set; }
	public double alfa { get; private set; }
	public double Mw { get; private set; }
	public double Kf { get; private set; }
	public double a { get; private set; }
	public double Kl { get; private set; }
	public double amax { get; private set; }
	public List<double> analyzyCasy = null;
	public List<double> analyzyHodnoty = null;
	…
}
public class Model
{
	private List<Slozeni> slozeni;
	…
}
```

Kód 2: Ukázka nového přístupu k datům složení

Provedené úpravy byly implementovány zejména pro odstranění výše popsaných vlastností. Díky těmto úpravám již nedochází ke konfliktům mezi vlákny ani k nepředpokládanému chování při zásahu uživatele do grafického rozhraní během výpočtu. Navíc byla snaha udělat kód přehlednější a rozšiřitelnější.

2.2 Popis algoritmu pro výpočet výsledků

Jak bylo popsáno v kapitole 1.1.2, je metoda výpočtu „chemie vždy“ časově velmi náročná a je třeba ji optimalizovat. Jeden způsob časové optimalizace byl již v programu proveden dříve. Tím bylo vytvoření skriptu SkriptVSE, který zajišťuje postupné počítání celé kolony v jednom skriptu místo toho, aby se pro každou buňku zvlášť volal skript a tím pádem by se pokaždé spouštěl externí program React.

Tato práce se zabývá druhým způsobem, kterým se dá časová náročnost snížit. Myšlenka rozšíření programu je následující. Do programu Transport bude implementováno, kromě již existujícího propojení s programem React, propojení s databází. Toto propojení by mělo bránit opakovanému volání programu React a spouštění chemického výpočtu a tím snížit celkovou dobu výpočtu programu.

Nový způsob optimalizace, dále také zvaný jako rozhraní Konnektor, by měl fungovat následujícím způsobem. Po provedení transportní části výpočtu programem Transport bude Konnektor pro každou buňku rozhodovat, zda budou výsledky reakční části výpočtu načteny z databáze nebo budou vypočteny programem React.

Přesněji to znamená, že Konnektor pro každou buňku kolony, po provedení transportní části, provede následující operace. Nejprve se podívá do databáze výsledků, jestli už buňka se stejným nebo dostatečně podobným složením není uložena v databázi. V případě, že je nalezen vhodný záznam, je použit k výpočtu výsledku. V případě, že žádný vhodný výsledek není v databázi nalezen, je pro zkoumanou buňku spuštěn program React.

Ke spuštění programu React je potřeba vytvořit příslušný skript. Skript je generován rozhraním Konnektor na základě šablony „ScriptSablona.rea“. Tato šablona musí být vytvořena uživatelem před zahájením výpočtu. Definuje specifické parametry modelu jako například informace o povolených a zakázaných minerálech nebo o teplotě modelu. V šabloně jsou také vyznačena místa, na která má Konnektor doplnit informace o spéciích, minerálech a jejich kinetických vlastnostech ve zkoumané buňce. Rozhraní Konnektor rovněž doplní na konec šablony předpis pro zápis výsledků výpočtu a vygeneruje výsledný skript. Šablona, na rozdíl od skriptu použitého v předchozí verzi programu, neobsahuje žádné informace o čtení a zápisu dat. O to se stará rozhraní Konnektor. Je tedy z uživatelského hlediska přehlednější než skript SkriptVSE.

Skript, který je pomocí šablony vygenerován, je následně spuštěn programem React. Po dokončení výpočtu rozhraní Konnektor načte výsledky výpočtu. Poté se provede výpočet vyhledávacího klíče uložení výsledků do databáze. Do databáze se ukládá vyhledávací klíč, vstupní data a vypočtená výstupní data. Konkrétně se uložení v databázi bude věnovat kapitola 2.4.

Po uložení dat do databáze je zkoumána další buňka. Před voláním programu React je opět nejprve prohledána databáze. Prohledávání v databázi se provádí na základě porovnávání vstupních dat, se kterými by byl volán program React. V případě, že je v databázi nalezen výsledek, který odpovídá zadané toleranci, je použit k výpočtu výsledku záznam načtený z databáze. Výsledek je rozhraním Konnektor aproximován pomocí takto načteného záznamu. Tato aproximace je provedena ke snížení chyby, která vznikla kvůli použité toleranci. Podrobně se této funkcionalitě bude věnovat kapitola 2.5.

Načtení dat z databáze a aproximace výsledků nahrazuje generování skriptu a volání programu React. Předpokladem je, že výpočet pomocí databáze bude podstatně rychlejší než volání externího programu a provádění numerického výpočtu. Použití rozhraní Konnektor, které kombinuje databázi a program React, by tedy mělo být časově efektivnější, než kdyby program Transport volal program React pro každou buňku.

Databáze slouží k ukládání průběžných výsledků během výpočtu, které by měly nahradit výpočet programem React. Do databáze se tedy budou ukládat formátované výsledky tak, aby po jejich přečtení program Transport dostal stejný formát výsledku nezávisle na tom, zda byl výsledek získán z databáze nebo vypočten pomocí programu React. Vzhledem k tomu, že výpočet probíhá pro každou buňku zvlášť, bude i v databázi jeden řádek tabulky reprezentovat jednu buňku kolony nezávisle na tom, ve kterém časovém kroku je počítána a na jakém místě je v koloně umístěna.

Výsledky výpočtu mohou být trojího druhu. První skupinu tvoří koncentrace jednotlivých složek, které jsou v roztoku zkoumané buňky obsaženy. Druhou skupinou pak tvoří množství jednotlivých minerálů, které jsou ve zkoumané buňce obsaženy. Samostatnou skupinu pak tvoří hodnota aktivity H^+, respektive pH.

Je třeba zdůraznit, že není možné kombinovat rozhraní Konnektor se způsobem, který využívá skript SkriptVSE. Jde o dva různé způsoby časové optimalizace, které jsou oba

postaveny na tom, že brání opakovanému volání programu React, každý to však dělá jiným způsobem. SkriptVSE zajistí, že je celá kolona sestavena do jednoho skriptu, který je pak hromadně spuštěn. Rozhraní Konnektor přistupuje ke každé buňce zvlášť a určuje, zda bude program React volán nebo ne.

Není tedy stoprocentně zajištěno, že nová verze programu bude ve všech případech efektivnější než původní verze programu. Z toho důvodu bylo rozhodnuto, že nová verze programu bude podporovat obě verze optimalizace, aby si uživatel mohl vybrat tu, která mu bude více vyhovovat. Základním předpokladem je, že každá z verzí optimalizace výpočtu bude rychlejší, než kdyby program Transport volal externí program React zvlášť pro každou buňku.

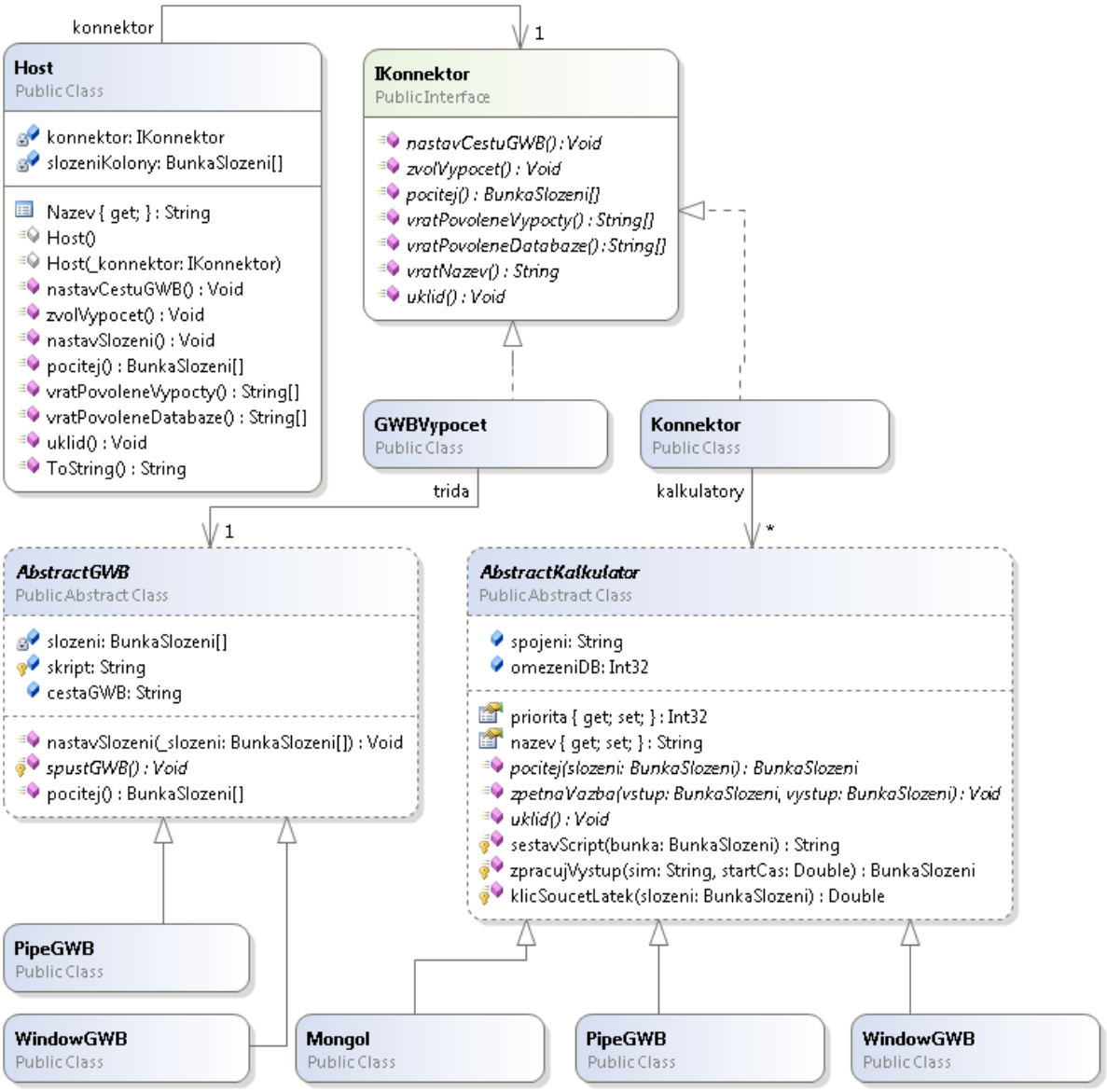
2.3 Struktura rozhraní Konnektor

Při návrhu rozhraní Konnektor a jeho připojení k programu Transport byl brán ohled na další rozšiřitelnost. Za prvé bylo nutné počítat s tím, že dva způsoby optimalizace, které jsou zatím v programu Transport implementovány, nejsou jediné dva možné a mohou být navrženy další. Struktura by tedy měla být navržena tak, aby při implementaci případného nového způsobu optimalizace nebylo nutné znovu do programu zasahovat tak zásadním způsobem, jako je tomu v této práci

Podobně je třeba zohlednit fakt, že program React není jediný existující program pro výpočet chemické části a může také být nahrazen jiným, například programem PhreeqC, se kterým program Transport pracoval ve svých prvních verzích. Dále je třeba počítat s tím, že databáze, která byla v této práci vybrána, může být nahrazena, případně doplněna, nějakou jinou.

Na obrázku 4 je zobrazen zjednodušený diagram tříd rozhraní Konnektor a jeho připojení k programu Transport. Je zde uvedena pouze základní struktura rozhraní. Kompletní diagram je uveden na přiloženém CD.

Struktura byla navržena tak, že každý druh optimalizace reakční části výpočtu je brán jako plugin k programu Transport. Při spuštění programu Transport je prohledán adresář, určený pro pluginy a jsou načteny informace o tom, které dynamicky linkované knihovny (DLL) jsou k dispozici. Pro každou načtenou knihovnu je vytvořena instance třídy Host. Uživatel si pak v grafickém rozhraní vybere plugin, který chce pro výpočet použít. Dále program Transport pracuje pouze s instancí, která odpovídá výběru uživatele.



Obrázek 4: Diagram tříd rozhraní Konnektor

Jako plugin je programem Transport chápána každá třída implementující rozhraní IKonnektor, jejíž DLL je uložena v adresáři Plugins, který je podadresářem hlavního adresáře, ve kterém je program Transport nainstalován.

V současné době existují dvě třídy implementující rozhraní IKonnektor. Třída GWBVypocet reprezentuje původní způsob spouštění reakční části programu, tedy výpočet prováděný za pomoci volání skriptu SkriptVSE. Třída Konnektor reprezentuje nový způsob výpočtu. To znamená volání dotazu na výpočet reakční části výpočtu pro každou buňku zvlášť a výběr, zda bude výpočet načten z databáze nebo vypočten programem React.

Pokud se uživatel rozhodne využívat plugin SkriptVSE, musí ještě před spuštěním výpočtu zvolit, jakým způsobem má být volán program React, zda s grafickým rozhraním nebo bez něj. K tomu slouží abstraktní třída AbstractGWB a potomci této třídy.

Třída AbstractGWB má na starost uložení dat do souborů, které bude skript, později volaný metodou spustGWB, načítat a se kterými bude dále pracovat. Jednotlivé třídy, dědící od této abstraktní třídy, se pak liší pouze implementací metody spustGWB a to tak, aby byla volána buď s grafickým rozhraním (třída WindowGWB) nebo bez něj (třída PipeGWB).

Podobná logika, jako byla nyní popsána pro plugin SkriptVSE, je použita i pro plugin Konnektor. Na rozdíl od předchozího pluginu má hlavní výpočetní třída, v tomto případě třída Konnektor, více instancí realizujících reakční výpočet a stará se o logiku, která instance bude ke konkrétnímu výpočtu zavolána.

Třídy dědící od abstraktní třídy AbstractKalkulator se starají o vlastní provedení výpočtu, tedy buď načtení dat z databáze (třída Mongol) nebo výpočet pomocí programu React s grafickým rozhraním (třída WinodwGWB) nebo výpočet pomocí programu React bez grafického rozhraní (třída PipeGWB).

Třída Konnektor obsahuje obecně několik instancí třídy AbstractKalkulátor. V současné době je implementováno rozhraní pouze pro dva druhy výpočtu. Třída Konnektor bude mít v současné verzi programu maximálně jednu instanci pro výpočet pomocí databáze a právě jednu pro výpočet pomocí programu React.

Do budoucna se počítá s tím, že tato funkcionalita může být rozšířena například o využití více databází nebo o částečnou paralelizaci výpočtu. Z toho důvodu má každá instance svou vlastní prioritu. Na začátku výpočtu si třída Konnektor vytvoří instance, podle volby uživatele, a setřídí je podle priority. Když přijde požadavek na provedení jednoho výpočtu, zkouší třída Konnektor postupně všechny výpočty podle priority, dokud nedostane výsledek. V případě, že se nějaké instanci podařilo dosáhnout výsledku, jsou o tom informovány všechny instance s vyšší prioritou a je vrácen výsledek. V případě, že ani instance s nejnižší prioritou nedokáže úlohu spočítat, je program ukončen chybou.

Konkrétně to znamená, že třída Konnektor má jednu instanci s vysokou prioritou, tou je výpočet pomocí databáze, a jednu instanci s nízkou prioritou, tou je výpočet pomocí programu React. Nejprve se provede pokus o načtení výsledku z databáze a v případě neúspěchu se teprve

provede zavolání programu React, uložení jeho výsledků do databáze a vrácení výsledku. Druh databáze a druh výpočtu programem React je vybrán uživatelem.

2.4 Návrh databáze

Tato kapitola je věnována výběru databáze, která bude v pluginu Konnektor použita. Struktura programu byla uspořádána tak, aby umožňovala jednoduchou implementaci jiné databáze, než která byla použita v této práci. Zde jsou popsány důvody, které vedly k volbě databáze, pro tuto verzi programu Transport.

Jak bylo popsáno v kapitole 2.2, jeden řádek některé z tabulek databáze reprezentuje jednu buňku kolony. Každý ze sloupců některé z tabulek pak reprezentuje jeden z výsledků výpočtu, tedy buď koncentraci některé ze složek, nebo obsah jednoho z minerálů nebo pH vybrané buňky. Kolik bude mít databáze tabulek a co budou reprezentovat závisí na výběru konkrétní databáze.

Kromě samotných výsledků je třeba do databáze ukládat vyhledávací klíč. Bylo by vhodné, kdyby vyhledávací klíč nebyl vybrán nahodile, ale nějakým způsobem reprezentoval obsah buňky. Výběru vyhledávacího klíče je věnována kapitola 2.4.2.

Dále kromě samotných výsledků je do databáze ukládáno zadání výpočtu, se kterým byly výsledky získány. Důvodem je, že při vyhledávání pomocí vyhledávacího klíče jsou výsledky vyhledávány s určitou toleranci. Následně je provedena kontrola všech vstupů, taktéž s tolerancí, a je vybrán záznam, který je nejbližší požadovanému. Aby byla chyba, která je způsobena tolerancí, co nejmenší, je ještě na základě porovnání vstupních a výstupních hodnot provedena aproximace výsledku.

2.4.1 Výběr databáze

Jak bylo popsáno v kapitole 1.2, v návrhu byly uvažovány jak relační databáze, tak NoSQL databáze. Zde budou již detailně rozebrány výhody a nevýhody těchto druhů databází pro konkrétní druh úloh, které Konnektor používá. Od každého z typů databází byl vybrán jeden zástupce, který bude porovnáván. Za relační databáze je to databáze MS SQL a to zejména z důvodu snadné implementace a kompatibility vzhledem k jazyku C# a frameworku .NET, ve kterém je program Transport implementován. Za NoSQL byl vybrána databáze MongoDB. Tato databáze byla vybrána na základě předchozích zkušeností s ní a na základě toho, že splňuje všechny podmínky, které jsou na databázi v této úloze kladeny.

Jedním ze základních parametrů databáze, která má být použita, je její rychlost. Předpokladem pro správné fungování rozšíření je totiž fakt, že výpočet pomocí databáze bude rychlejší než volání externího programu a výpočet pomocí něj. Na druhou stranu je však třeba vzít úvahu, že záznamy do databáze jsou ukládány pro každý výpočet zvlášť, tedy před zahájením jiného výpočtu v programu Transport je databáze promazána. Dále i v rámci jednoho výpočtu mohou záznamy v databázi „stárnout“. Tedy mohou existovat záznamy, které v databázi již nejsou potřeba a mohou být průběžně promazávány.

Je tedy možné předpokládat, že databáze bude obsahovat relativně malé množství záznamů a že tedy bude rychlejší než výpočet pomocí programu React. To, jak velikost databáze ovlivňuje průběh výpočtu je zajímavá úloha k testování a bude zkoumána v kapitole 3.

Omezení délky databáze má za důsledek fakt, že délka výpočtu pomocí databáze bude řádově stejně dlouhá, ať už je databáze relační nebo NoSQL. Tento parametr, přestože je velmi důležitý, tedy nijak zásadně neovlivňuje to, která konkrétní databáze bude nakonec vybrána. Bylo by sice možné testovat rychlost různých databází pro tento druh úloh, ovšem není to cílem této práce.

Druhým parametrem, který je třeba brát v úvahu, je fakt, že informace o tom, jaké složky a jaké minerály se v koloně mohou vyskytovat během výpočtu, jsou známy až po spuštění programu Transport. Bylo by tedy velmi obtížné a neefektivní vytvořit jednu univerzální databázi, která by mohla sloužit pro všechny možné úlohy. Mnohem efektivnější a jednodušší je vytvořit tabulky databáze až podle toho, jaké je zadání řešené úlohy a po skončení výpočtu celou databázi opět smazat.

S tím také souvisí návrh vnitřní struktury databáze. První možností je vytvořit jednu velkou tabulku, která bude obsahovat všechny informace. Každý řádek této tabulky pak bude reprezentovat přesně jednu buňku. Každý sloupec pak buď koncentraci složky ze zadání nebo koncentraci složky z výsledku výpočtu respektive množství minerálu ze zadání nebo množství minerálu z výsledku výpočtu.

Druhou možností je vytvořit pro každou dvojici zadání – výsledek jednu tabulku. Každá tabulka by pak obsahovala tři sloupce jeden by byl vyhledávací klíč, který by byl zároveň cizím klíčem do nějaké spojující tabulky. Další sloupec by pak reprezentoval hodnotu ze zadání, třetí sloupec pak hodnotu výsledku. Přidání nového minerálu by pak znamenalo přidání nové tabulky

do databáze. Tento způsob více odpovídá relačnímu uložení databáze, pro účely, pro které má sloužit je však zbytečně složitý a nepřináší žádnou výhodu oproti způsobu již popsanému.

Vzhledem k tomu, že ani jeden z výše popsaných parametrů pro výběr databáze nerozhodl jednoznačně pro jeden nebo druhý typ databáze, je třeba přistoupit k porovnání konkrétních vybraných databází, tedy databáze MS SQL a MongoDB.

Vzhledem k tomu, že se způsob ukládání do jedné velké tabulky ukázal jako nejefektivnější, nejsou od databáze požadovány žádné jiné operace než vložení záznamu do tabulky a jeho vyhledání. Uživateli tedy stačí jednoduchá databáze se snadnou instalací, která nebude zabírat mnoho místa na disku. Tento parametr lépe splňuje databáze MongoDB.

Druhým z vedlejších parametrů je parametr, který usnadňuje implementaci. Je to parametr pojmenování sloupců. Vzhledem k tomu, že složky roztoku mohou být buď kationty nebo anionty a jsou tedy kromě prvků, ze kterých jsou složeny, označeny také znaky plus nebo mínus, je vhodné, kdyby na pojmenování sloupců v databázi byla kladena co nejmenší omezení. Databáze MongoDB, vzhledem k tomu, že data ukládá ve formátu BSON, odvozeného do formátu JSON, tedy jako kolekci dvojic klíč – hodnota, neklade na pojmenování sloupců téměř žádná omezení. Databáze MS SQL považuje některé znaky za speciální a nemohou se v názvech sloupců objevovat.

Třetí parametr, který je důležitý zejména pro účely ladění programu, je snadný přístup k datům v databázi i odjinud než z programu Transport. Pro uživatele programu může také být tato informace zajímavá v případě, že je model vypočítán jinak, než bylo očekáváno. Uživatelsky příjemnější prostředí pro manipulaci s databází nabízí databáze MS SQL.

Protože žádný z uvedených argumentů jednoznačně nerozhodl pro jednu nebo druhou databázi, byla vybrána databáze MongoDB a to na základě předchozích zkušeností s touto databází.

2.4.2 Výběr vyhledávacího klíče

Nezávisle na druhu databáze bylo nutné vybrat vhodný vyhledávací klíč, pomocí kterého budou záznamy z databáze vyhledávány. Popis algoritmu, jak bude vyhledávání probíhat, byl popsán v kapitole 2.2.

Bylo rozhodnuto, že pro efektivní vyhledávání v databázi by neměl být klíč vybrán nebo

generován náhodně, ale měl by nějakým způsobem reprezentovat data, která se na řádku nacházejí. Vzhledem k tomu, že při vyhledávání v databázi jsou známy pouze vstupní hodnoty, tedy koncentrace složek a množství minerálů, obsažených ve zkoumané buňce, může být klíč vypočten pouze z těchto hodnot.

Jako klíč byl tedy vybrán prostý součet koncentrací složek, obsažených v buňce. Minerály do vyhledávacího klíče nebyly zahrnuty. Přestože se jedná o velice jednoduchý hash, má i další významy, které vyplývají z povahy parametrů, ze kterých byl vypočten.

Z hlediska hydrochemického se součet koncentrací anorganických látek rozpuštěných ve vodě označuje jako celková mineralizace. Celková mineralizace patří mezi jeden ze základních ukazatelů kvality vody a odpovídá reálnému složení dané vody. Podrobnější informace o významu a měření celkové mineralizace je možné nalézt například v článku[5], ze kterého tento odstavec vychází.

Z hlediska lineární algebry je možné přistupovat ke vstupním datům jako k vektoru. Velikost každé vstupní hodnoty, respektive koncentrace, pak představuje velikost jedné z dimenzí vektoru. Součet všech složek pak představuje normu vektoru, protože vzhledem k vlastnostem, které vstupní data mají, jsou splněny všechny podmínky normy vektoru.

První vlastností normy vektoru je fakt, že nulovou normu má pouze nulový vektor. Ostatní vektory mají normu kladnou. Vzhledem k tomu, že každá složka vektoru představuje koncentraci některé anorganické látky a koncentrace látek jsou nezáporné, jejich součet musí být také nezáporný a nulový pouze v případě, že jsou všechny koncentrace nulové. Další vlastnosti normy, tedy homogenita a trojúhelníková nerovnost, jsou také splněny díky nezápornosti všech složek vektoru.

2.4.3 Omezení databáze

Data v databázi mohou v průběhu výpočtu stárnout. Znamená to, že ke konci výpočtu databáze může obsahovat data ze začátku výpočtu, která již nejsou aktuální. Vzhledem k tomu, že jsou data z databáze načítána s tolerancí, je možné, že tato starší data mohou výsledky výpočtu ovlivnit. Proto bylo rozhodnuto, že databáze bude omezená na určitý maximální počet záznamů. V případě, že by při zapsání záznamu do databáze byl přesažen maximální počet záznamů v databázi, bude programem odstraněn z databáze nejstarší záznam. Na kolik záznamů má být databáze omezena, aby byla stále efektivní, bude zkoumáno v kapitole 3.

2.5 Tolerance výsledků a jejich aproximace

Rozhraní Konnektor vyhledává výsledky reakční části v databázi se zadanou tolerancí. Jaký má velikost této tolerance vliv na průběh výpočtu bude zkoumáno v kapitole 3. Rozhraní Konnektor vyhledává výsledky v databázi ve třech krocích. Nejprve dojde k vyhledání pomocí klíče, následuje porovnání vstupů a nakonec aproximace výsledku.

V prvním kroku algoritmu se vypočte celková mineralizace buňky, která je zkoumána. Tato hodnota je porovnána s databází a z databáze jsou načteny všechny záznamy, které mají hodnotu vyhledávacího klíče v okolí hodnoty vypočtené celkové mineralizace daném zadanou tolerancí. Tyto hodnoty představují složení buněk, které byly již dříve vypočteny a uloženy do databáze. Vyhledané záznamy jsou vráceny rozhraní Konnektor pro další filtraci. V případě, že není nalezen žádný takový záznam, je tento algoritmus ukončen a je zavolán program React.

V druhém kroku jsou postupně porovnány koncentrace všech složek vstupní buňky. Z kolekce buněk, které jsou načteny z databáze, jsou následně vyřazeny buňky, jejichž hodnoty koncentrací složek se liší od hodnot koncentrací složek v buňce, která je zkoumána, o více, než je zadaná tolerance. V případě, že v kolekci nezbyde žádný záznam, je algoritmus ukončen a je zavolán program React pro výpočet reakční části výpočtu.

V případě, že v kolekci zbyla alespoň jedna buňka, je vybrána buňka, která je zkoumané buňce nejpodobnější. Rozhodují zde dva faktory. Prvním z nich je obsah minerálů. V případě, že porovnávaná buňka minerály obsahuje, nezávisle na jejich množství, jsou z kolekce vyřazeny všechny buňky, které minerály neobsahují a obráceně.

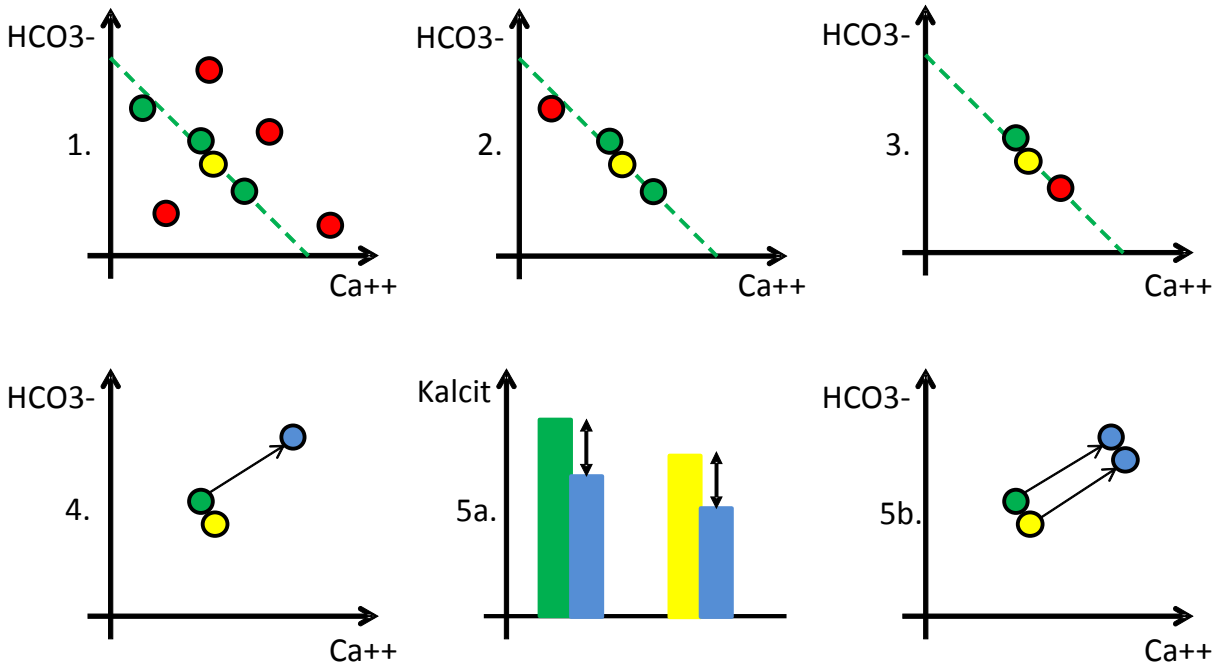
Po takto provedené filtraci je porovnán poslední faktor a to je skutečná blízkost vstupů. Z načtené kolekce buněk je vybrána ta nejbližší k buňce zkoumané. Pokud je buňka brána jako bod v prostoru, kde každá koncentrace určuje jednu dimenzi prostoru, pak je vybrána buňka, která je v tomto prostoru nejblíže buňce zkoumané.

Nyní už je pouze porovnávána zkoumaná buňka s takto vybraným záznamem z databáze, který má význam dříve vypočteného složení buňky, a jsou aproximovány výstupní hodnoty zkoumané buňky. Aproximace je provedena následovně. Nejprve jsou přečteny výstupní koncentrace ze záznamu získaného z databáze a jsou označeny jako výstupní koncentrace zkoumané buňky. Následně jsou tyto koncentrace upraveny. Ke každé výstupní koncentraci zkoumané buňky je přičtena hodnota rozdílu vstupní koncentrace zkoumané buňky a vstupní

koncentrace získané ze záznamu načteného z databáze. Stejný postup je proveden i pro hodnoty množství minerálů v buňkách.

Zjednodušeně řečeno to znamená, že bylo rozpuštěno (respektive sraženo) určité množství minerálu, které reprezentuje rozdíl vstupní a výstupní hodnoty minerálu a je konstantní pro načtený záznam z databáze a zkoumanou buňku. Při tomto rozpouštění (respektive srážení) došlo k tomu, že v roztoku, kterým je buňka naplněna, přibylo (respektive ubylo) určité množství koncentrace složek. Hodnota, o kterou je každá z koncentrací složek změněna, je opět v rámci jedné koncentrace konstantní pro načtený záznam z databáze a buňku zkoumanou.

Průběh aproximace je schematicky znázorněn na obrázku 5. Je zde znázorněno rozpouštění kalcitu za vzniku složek Ca^{++} a HCO_3^-. Každá ze složek je brána jako jedna dimenze vektoru. Body na obrázku představují složení některé buňky.



Obrázek 5: Schematické znázornění aproximace výsledků

Na části obrázku, která je znázorněna číslem 1 je vyobrazen první krok filtrování. Žlutý bod jsou vstupní koncentrace zkoumané buňky. Zelená čára označuje všechny body se stejnou normou vektoru, jako má zkoumaná buňka. Zelené body jsou koncentrace buněk, které byly dříve spočítány a nyní byly získány z databáze, které mají normu stejnou jako zkoumaná buňka nebo v rámci použité tolerance. Červené body jsou buňky, které tuto podmínku nesplňují a budou v tomto kroku filtrací vyřazeny.

Na druhé části obrázku (označená číslem 2) je znázorněn druhý krok filtrování, tedy porovnání všech vstupních koncentrací získaných ze záznamů, které byly načteny z databáze, a vyřazení buněk, které tuto podmínku nesplňují. Barevná interpretace je stejná, jako na předchozím obrázku, tedy žlutý bod reprezentuje zkoumanou buňku, zelené body reprezentují buňky, které podmínku splňují a jsou dostatečně blízko zkoumané buňky a červený bod reprezentuje bod, který podmínku nesplnil a bude v tomto kroku vyřazen.

Na třetí části obrázku je vybrána buňka, která je nejbližší zkoumané buňce. Opět do dalšího kroku zůstává žlutý bod jako zkoumaná buňka a zelený bod jako jediná buňka, která podmínku splňuje. Červený bod je v tomto kroku vyřazen.

Následně již dochází k samotné aproximaci výstupních hodnot. Na části 4 je možné vidět žlutý bod, jako vstupní hodnoty zkoumané buňky, zelený bod jako vstupní hodnoty buňky získané filtrací a modrý bod jako výstupní hodnoty buňky získané filtrací.

Obrázky 5a a 5b pak reprezentují samotný přepočet výstupních hodnot zkoumané buňky. Na obrázku 5a je vidět, že při přechodu ze zeleného bodu, který reprezentoval vstupní hodnoty buňky, do modrého bodu, který reprezentuje výstupní hodnoty buňky, ubylo množství kalcitu, které je označeno konstantou c. Stejné množství kalcitu ubylo i při přechodu ze žlutého bodu (vstupní hodnoty) do modrého bodu (výstupní hodnoty), které reprezentují zkoumanou buňku. Na obrázku 5b je pak vidět přepočet výstupních koncentrací složek buňky. Je vidět, že směr i velikost zobrazení, které znázorňuje přechod ze zeleného bodu do modrého, tedy ze vstupních koncentrací do výstupních vzhledem k zánamu načteného z databáze, jsou stejné jako směr a velikost zobrazení, které znázorňuje přechod ze žlutého bodu do modrého, tedy ze vstupních koncentrací do výstupních vzhledem ke zkoumané buňce. Postup, který je zde popsán zajišťuje, že při použité aproximaci výsledků je zachována látková bilance. Nezajišťuje však, že je zachována chemická rovnováha v aproximované buňce.

3 Rozbor dosažených výsledků

V této kapitole bude rozebrána efektivita vytvořeného rozhraní Konnektor na testovacím modelu. Zkoumány budou zejména výpočetní čas a velikost chyb, vzniklých aproximací výsledků, v závislosti na velikosti databáze a na použité toleranci. Výpočty budou prováděny na stejné koloně s různým počtem buněk a odpovídajícím časovým krokem tak, aby mohly být výsledy vzájemně porovnány. Všechny testy byly spouštěny na stejném počítači (procesor Intel Core i3 3240v 3,4 Ghz, 6 GB RAM, operační systém Windows 7 64-bit).

3.1 Popis testovacího modelu a parametrů spouštění

Jako testovací model byl vybrán model rozpouštění kalcitu. Je uvažována kolona válcového tvaru o délce 40 mm a průměru 7,9 mm. Kolona je vyplněna kalcitem o množství 0,1 milimolu na kilogram vody a roztokem, který je v rovnováze s tímto prostředím. Kinetické parametry kalcitu jsou zvoleny následovně. Povrch kalcitu je nastaven na 1000 cm^2/g a rychlostní konstanta na 5×10^{-9} mol/cm^2s. Teplota v koloně je nastavena na 25 °C.

Do kolony je vtláčena čistá voda, tím bude docházet k rozpouštění kalcitu na Ca^{++} a HCO_3^-. Budou tedy sledovány koncentrace těchto dvou složek, množství kalcitu, které zůstává v buňkách a změna pH. Voda protéká kolonou rychlostí 0,294 ml/min po dobu 5,5 min.

Tento model bude postupně spouštěn s různými parametry. Prvním parametrem, který se bude lišit je počet buněk kolony a časový krok výpočtu. Časový krok je volen v závislosti na počtu buněk kolony tak, aby bylo možné výsledky vzájemně porovnat. Model bude spouštěn postupně na 10 buněk s časovým krokem 0,05 minuty, dále na 50 buněk s časovým krokem 0,01 minuty a 100 buněk s časovým krokem 0,005 minuty.

Všechny výpočty, které budou spouštěny, budou pro výpočet reakční části používat metodu volání „chemie vždy“. Ostatní metody nejsou pro toto testování zajímavé, protože nejsou tolik časově náročné jako tato metoda. Metody spouštění reakční části byly popsány v kapitole 1.1.2. Program React bude spouštěn ve všech případech bez grafického rozhraní.

Výpočet bude volán celkem sedmi způsoby pro každý ze zvolených počtů buněk. Tyto způsoby se vzájemně liší tím, jaký je použit způsob optimalizace, případně jak je databáze omezena. První způsob, dále také označovaný jako NoDB, je způsob, při kterém se program React volá zvlášť pro každou buňku. Druhý způsob, dále označovaný jako SkriptVSE, je způsob,

který používá původní způsob časové optimalizace, tedy použití skript SkriptVSE.

Další způsoby využívají nového rozhraní Konnektor a liší se vzájemně od sebe maximálním počtem záznamů v databázi. Prvním způsobem je použití neomezené databáze, tedy databáze, které záznamy vůbec během výpočtu nepromazává, dále bude označován jako DBINF. Dále budou použity databáze o maximálních velikostech 25 záznamů (dále DB25), 50 záznamů (dále DB50), 75 záznamů (dále DB75) a 100 záznamů (dále DB100).

Třetím parametrem, nezávislým na parametrech počet buněk a způsobu výpočtu, je použitá tolerance při aproximaci výsledků. Tento parametr má smysl sledovat pouze u výpočtů, které používají databázi, nemá jej tedy smysl spouštět pro výpočty způsobem NoDB a SkriptVSE. Zvolené tolerance jsou 1%, 0,5% a 0,1%.

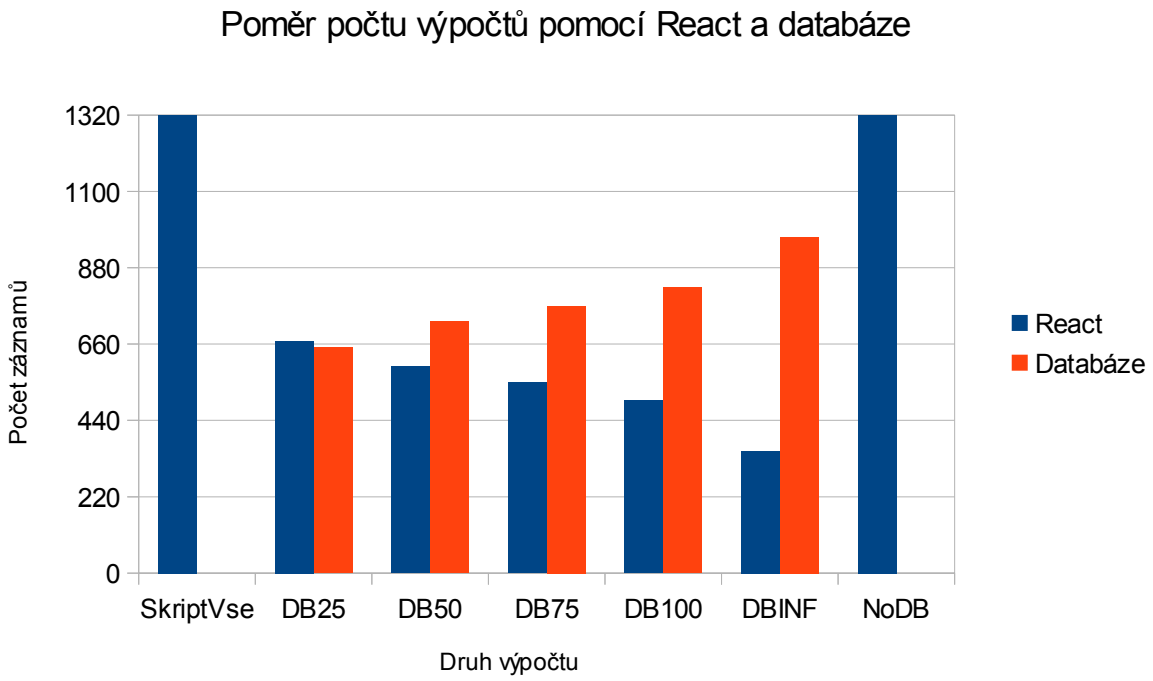
Celkem tedy vzniklo 51 kombinací vstupních parametrů, se kterými byl výpočet spouštěn. Následující kapitoly porovnávají vždy vybrané kombinace vstupních parametrů.

3.2 Testování modelu 10 buněk, 1% tolerance

V této kapitole jsou zvoleny parametry počet buněk na 10, odpovídající časový krok je 0,05 minuty, a je zvolena 1% tolerance při vyhledávání v databázi. Porovnáván je tedy třetí parametr, způsob výpočtu. Zkoumány jsou následující údaje.

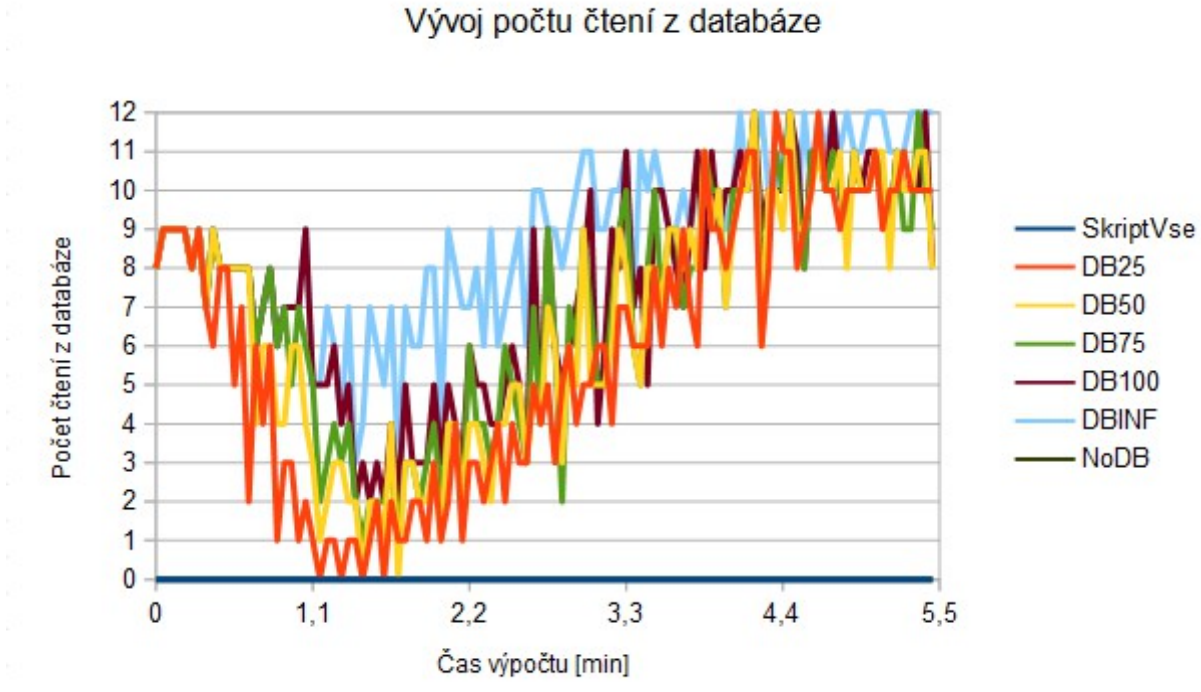
Prvním zkoumaným údajem je počet přečtených dat z databáze při zvoleném druhu výpočtu. Na obrázku 6 je uveden graf této závislosti. Během výpočtu bylo provedeno celkem 1320 výpočtů reakční části, protože bylo provedeno celkem 110 časových kroků a kolona obsahuje 10 buněk a dvě komory, jednu vstupní a jednu výstupní.

Na grafu je porovnáváno kolik výpočtů bylo provedeno programem React a kolik bylo načteno z databáze a aproximováno rozhraním Konnektor vzhledem k použitým druhým výpočtu. Počet výpočtů pomocí programu React současně odpovídá počtu zápisů do databáze, protože po každém výpočtu pomocí programu React je proveden zápis do databáze. U výpočtu druhu DBINF to zároveň ukazuje, kolik záznamů se na konci výpočtu v databázi nachází, konkrétně je to 352 záznamů. Druhý sloupec pak označuje počet čtení z databáze. Na grafu je vidět závislost, že čím je databáze více omezená, tím více záznamů je počítáno pomocí programu React.



Obrázek 6: Graf poměru výpočtu pomocí programu React a databáze – 10 buněk 1% tolerance

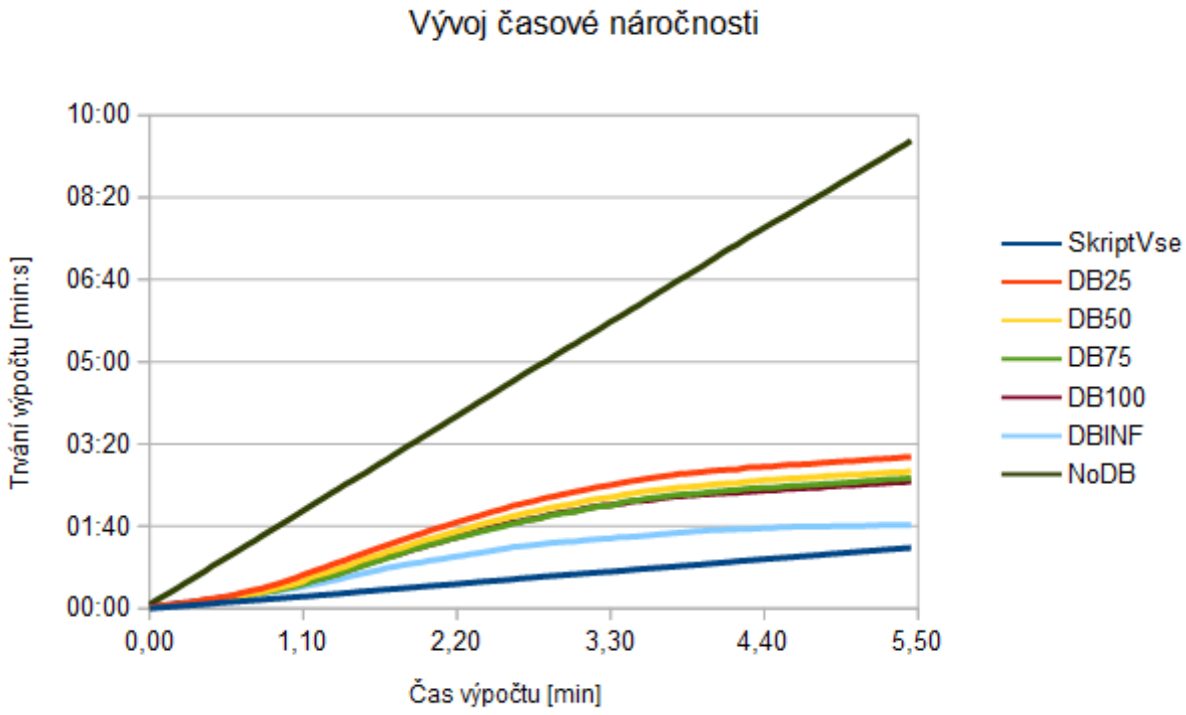
Následující graf, na obrázku 7, ukazuje, kolik záznamů bylo v jednotlivých časových krocích načteno z databáze a aproximováno rozhraním Konnektor. Počet výpočtů pomocí programu React je vždy doplňkem zobrazené hodnoty v daném časovém kroku do maximální hodnoty, tedy 12.



Obrázek 7: Graf vývoje počtu čtení z databáze – 10 buněk 1% tolerance

Přestože je graf málo přehledný ukazuje pro všechny druhy výpočtu následující trend. Na začátku výpočtu, kdy je kolona z většiny naplněna původním roztokem a přitékající roztok se dostal teprve do prvních buněk, jsou programem React vypočítány jen 3 – 4 buňky, ostatní záznamy jsou načteny z databáze. S přibývajícím časem se čelo kontaminace dostává dále do kolony a vznikají stále nové směsi. Proto je více záznamů počítáno programem React a méně záznamů je čteno z databáze. Znatelné je to zejména u druhu výpočtu DB25 po první minutě výpočtu, kdy jsou téměř všechny záznamy počítány programem React. V čase přibližně 1,3 minuty od začátku simulace totiž dojde k tomu, že se čelo kontaminace dostane až na konec kolony. Celá kolona nyní obsahuje ve všech buňkách směsi roztoku původního a roztoku přitékajícího a žádná z buněk již neobsahuje pouze roztok, kterým byla kolona naplněna původně. Postupně, jak je databáze plněna, dochází opět k čím dál častějšímu čtení dat z databáze místo spouštění programu React.

Dalším, velmi důležitým, sledovaným údajem je časová náročnost jednotlivých druhů výpočtu. Na grafu 8 je vyobrazena délka trvání výpočtu v závislosti na průběhu výpočtu. Na vodorovné ose je čas, ve kterém se výpočet nachází, na svislé ose pak skutečný čas, jak dlouho výpočet probíhal.



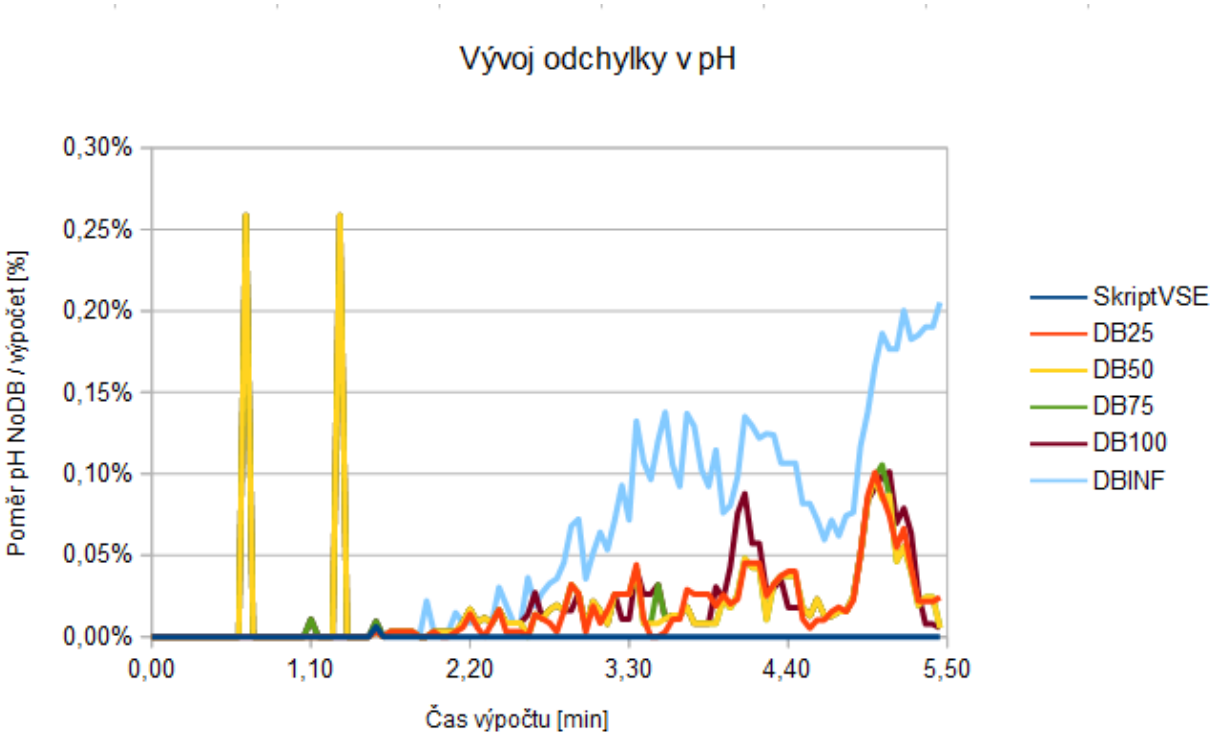
Obrázek 8: Graf vývoje časové náročnosti – 10 buněk 1% tolerance

Z grafu je možné vyčíst, že všechny druhy optimalizace, které byly použity, jsou podstatně rychlejší než výpočet NoDB, který volá program React pro výpočet všech buněk v reakční části

výpočtu. Pro použitou sadu parametrů je dále vidět, že doba trvání výpočtu je při použití libovolně omezené databáze přibližně stejná a také srovnatelná s původním způsobem optimalizace SkriptVSE.

Posledním sledovaným údajem je porovnání výsledků, které jsou jednotlivými druhy výpočtu získány. Vzhledem k tomu, že výstupy získané z databáze jsou aproximovány, mohou se lišit od výsledků, které by byly získány, kdyby byly počítány programem React. Následující grafy tedy budou srovnáním výsledků, které byly spočítány výpočtem NoDB s ostatními druhy výpočtu. Výsledky jsou zobrazeny jako procentuální odchylky zkoumané hodnoty daného druhu výpočtu od výpočtu NoDB. Odchylky jsou v absolutní hodnotě průměrovány přes všechny buňky v časovém kroku.

Na obrázku 9 je zobrazen vývoj odchylky pH výpočtů od výpočtu NoDB, který spouští program React pro každou buňku. Jak je vidět, všechny druhy výpočtu, které používají databázi, kromě výpočtu DBINF mají tuto odchylku srovnatelnou.

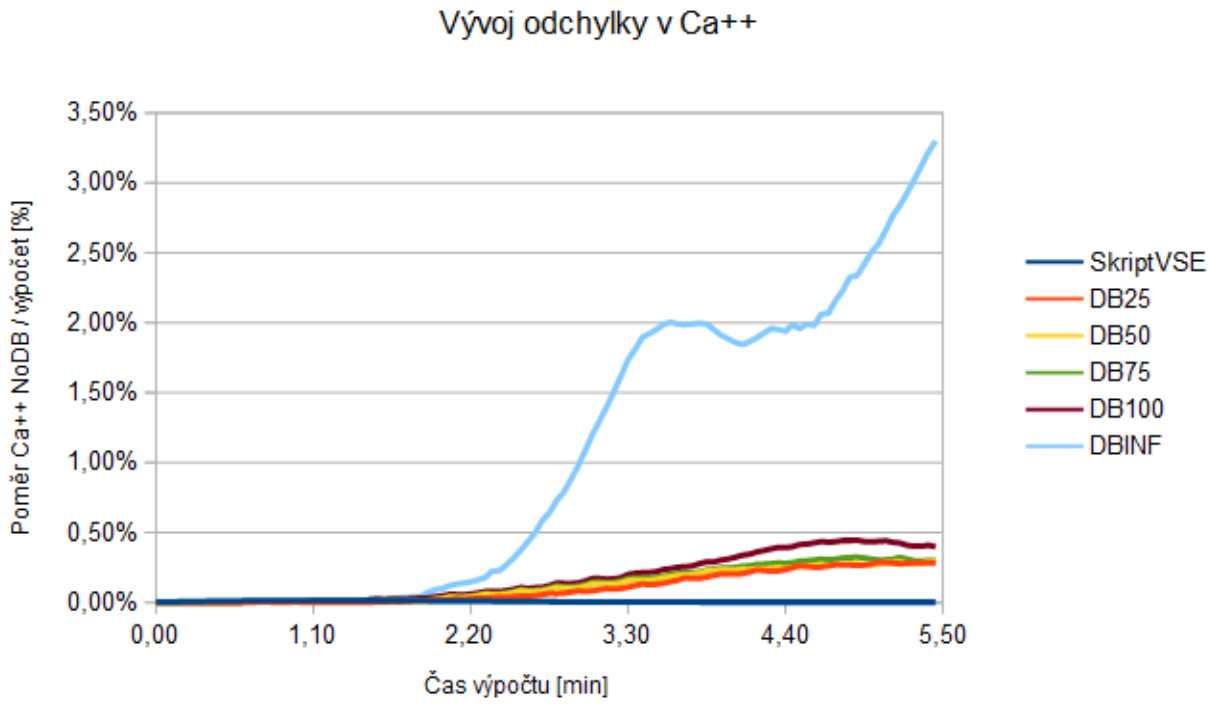


Obrázek 9: Graf vývoje odchylky v pH od výpočtu NoDB – 10 buněk 1% tolerance

Zajímavé jsou odchylky, které vznikly v časech 0,65 a 1,30. Přestože to z grafu není zřejmé, jsou tyto dvě odchylky stejné pro všechny druhy výpočtu, které používají databázi, tedy nejen pro DB50, jak to vypadá na grafu.

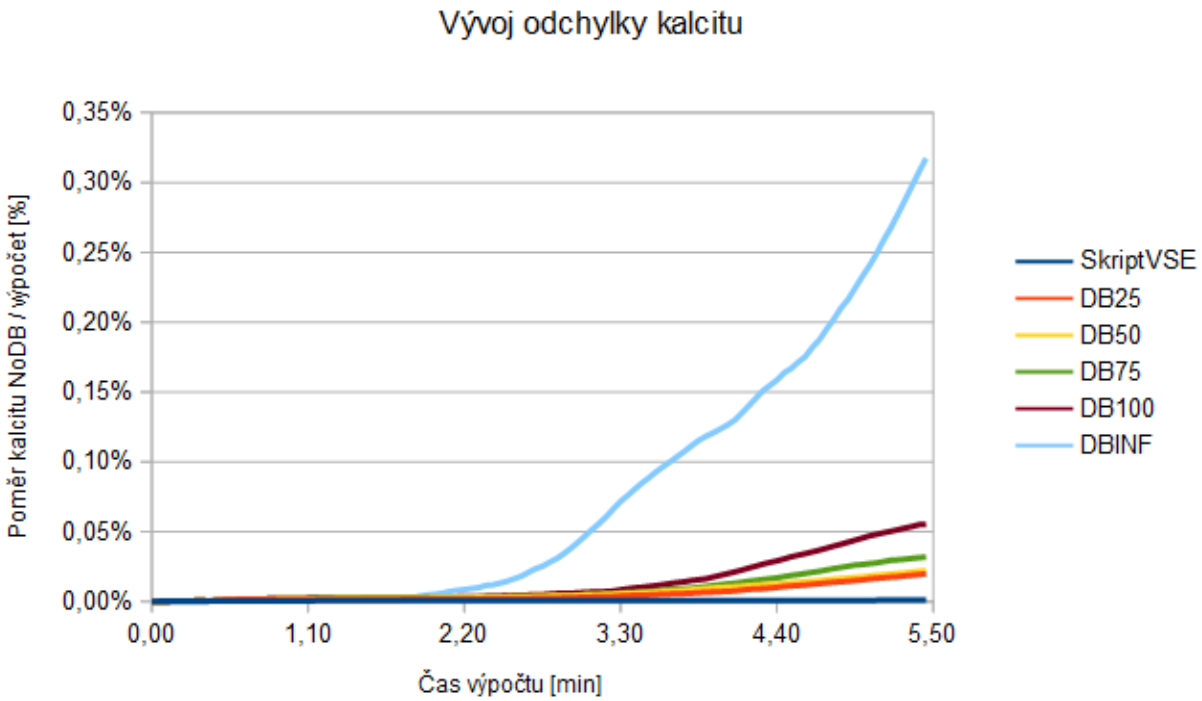
Takto extrémní výkyvy v odchylkách jsou způsobeny použitou aproximací, ale pouze pro hodnotu pH. Hodnota pH se totiž od ostatních hodnot liší tím, že ji není možné aproximovat postupem, která byl popsán v kapitole 2.5 a to z toho důvodu, že není známa vstupní hodnota pH zkoumané buňky. Místo algoritmu, který byl popsán, je výstupní hodnota pH zkoumané buňky pouze zkopírována z hodnoty pH buňky vybrané z databáze.

Na obrázku 10 je zobrazen graf vývoje odchylky v Ca^{++} různých druhů výpočtů od výpočtu NoDB. Zde je vidět, že odchylky výpočtů, které používají databázi, jsou srovnatelné. Jediný výpočet, který má jinou odchylku je výpočet DBINF, tedy výpočet, který používá neomezenou databázi. Z grafu je vidět, že odchylka tohoto druhu výpočtu, na rozdíl od ostatních, velmi rychle roste. Vzhledem k tomu, že je zachována látková bilance, jsou odchylky v HCO_3^- totožné s odchylkami v Ca^{++}.



Obrázek 10: Graf vývoje odchylky v Ca++ od výpočtu NoDB – 10 buněk 1% tolerance

Na obrázku 11 je zobrazen graf vývoje odchylky v kalcitu všech druhů výpočtu od výpočtu NoDB. Podobně jako u odchylky v Ca^{++} a v HCO_3^- je možné pozorovat rychlý nárůst odchylky u výpočtu, který používá neomezenou databázi.



Obrázek 11: Graf vývoje odchylky kalcitu od výpočtu NoDB – 10 buněk 1% tolerance

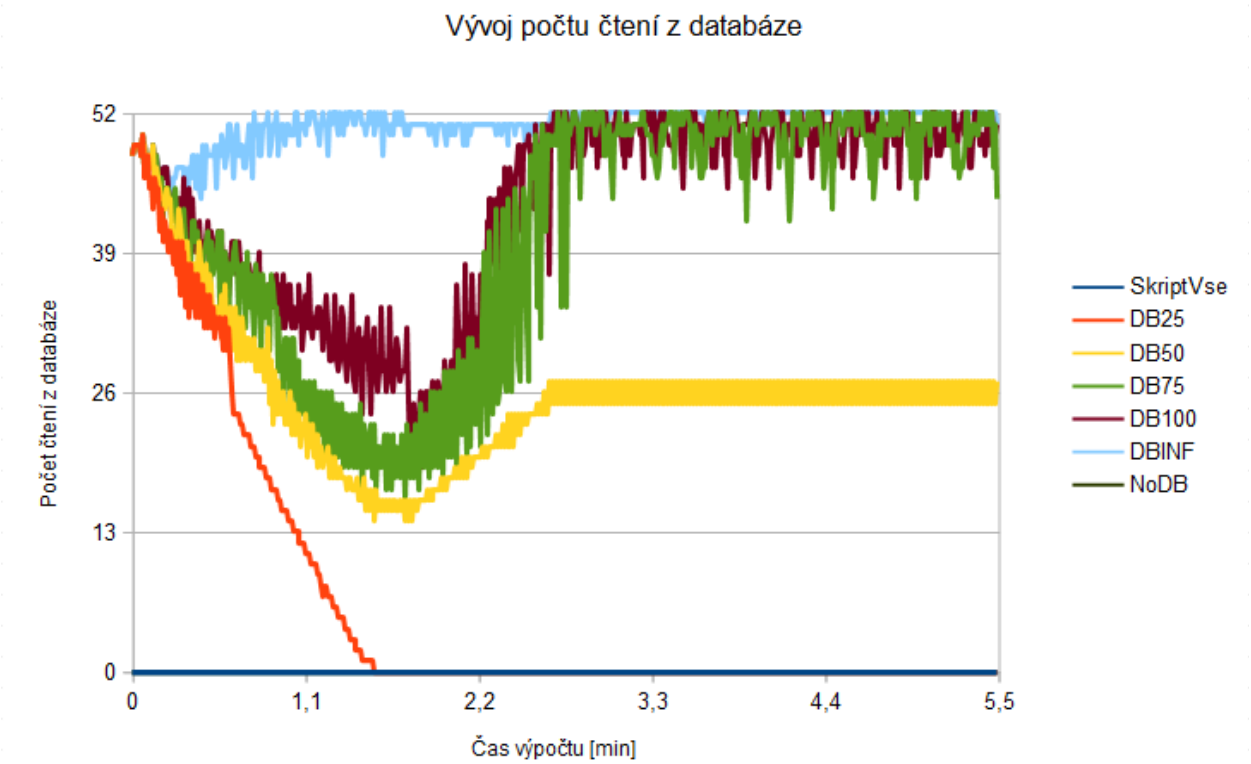
3.3 Testování modelu 50 buněk 1% tolerance

V této kapitole jsou zvoleny parametry počet buněk na 50, odpovídající časový krok je 0,01 minuty, a je zvolena 1% tolerance při vyhledávání v databázi. V textu práce budou nyní, na rozdíl od předchozí kapitoly, uvedeny jen některé grafy a analýzy. Kompletní seznam grafů je možné nalézt na přiloženém CD.

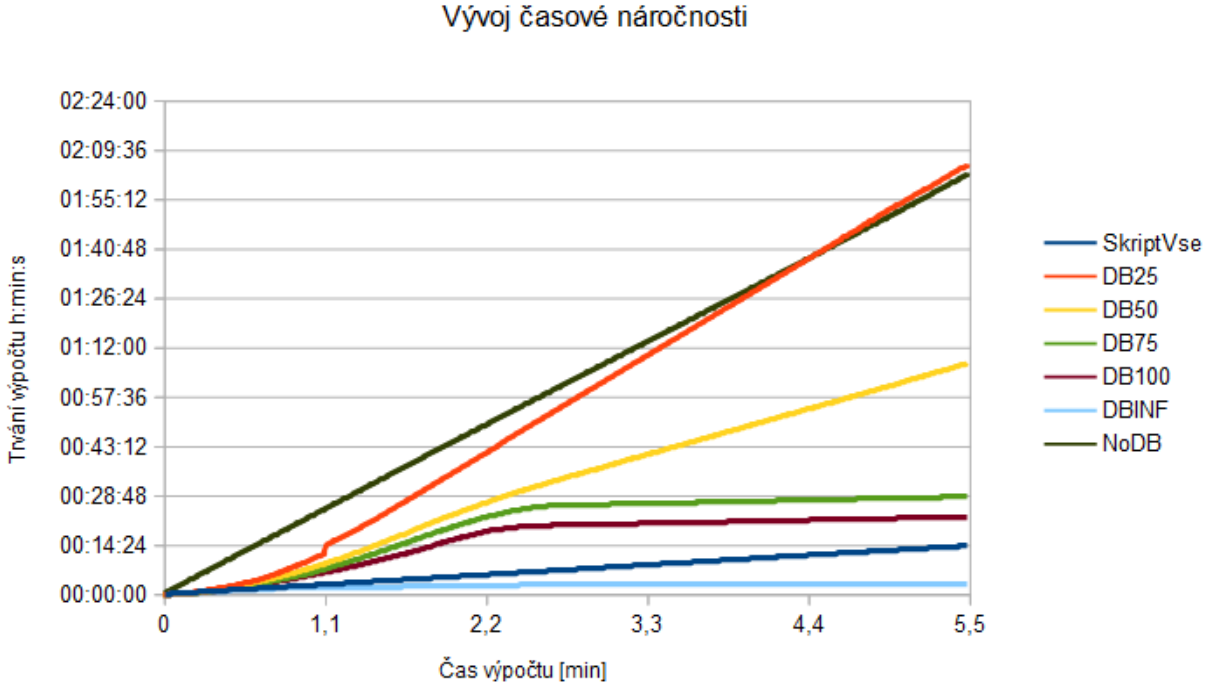
Na obrázku 12 je uveden graf vývoje počtu čtení z databáze v závislosti na průběhu výpočtu. Na rozdíl od odpovídajícího grafu, uvedeného v předchozí kapitole, je na tomto grafu více viditelný trend, který byl v předchozí kapitole popsán.

Je však možné si všimnout, že tento trend nesplňují výpočty typu DBINF a DB25. U výpočtu DBINF je možné vidět, že téměř všechny výpočty jsou nyní provedeny načtením z databáze a následnou aproximací. Je tedy možné očekávat velkou odchylku od výpočtu, který počítá reakční část výpočtu pouze za pomoci programu React. U výpočtu DB25 je možné si naopak všimnout, že po určitém čase již databáze není využívána vůbec. Je tomu tak zřejmě z toho důvodu, že je databáze příliš omezená a data jsou promazána dříve, než je možnost je využít. K tomuto jevu dochází přibližně v době, kdy se čelo kontaminace dostane na konec kolony a žádná z buněk již není naplněna pouze původním roztokem.

Na obrázku 13 je uveden graf vývoje časové náročnosti výpočtu. Zde je možné si všimnout souvislosti mezi počtem dat přečtených z databáze a dobou trvání výpočtu. Zatímco druh výpočtu, který prováděl všechny výpočty pomocí programu React, tedy výpočet NoDB a od času 1,53 i výpočet DB25, trvaly přibližně dvě hodiny. Výpočet, který používá prakticky pouze databázi, měl dobu trvání přibližně tři minuty.



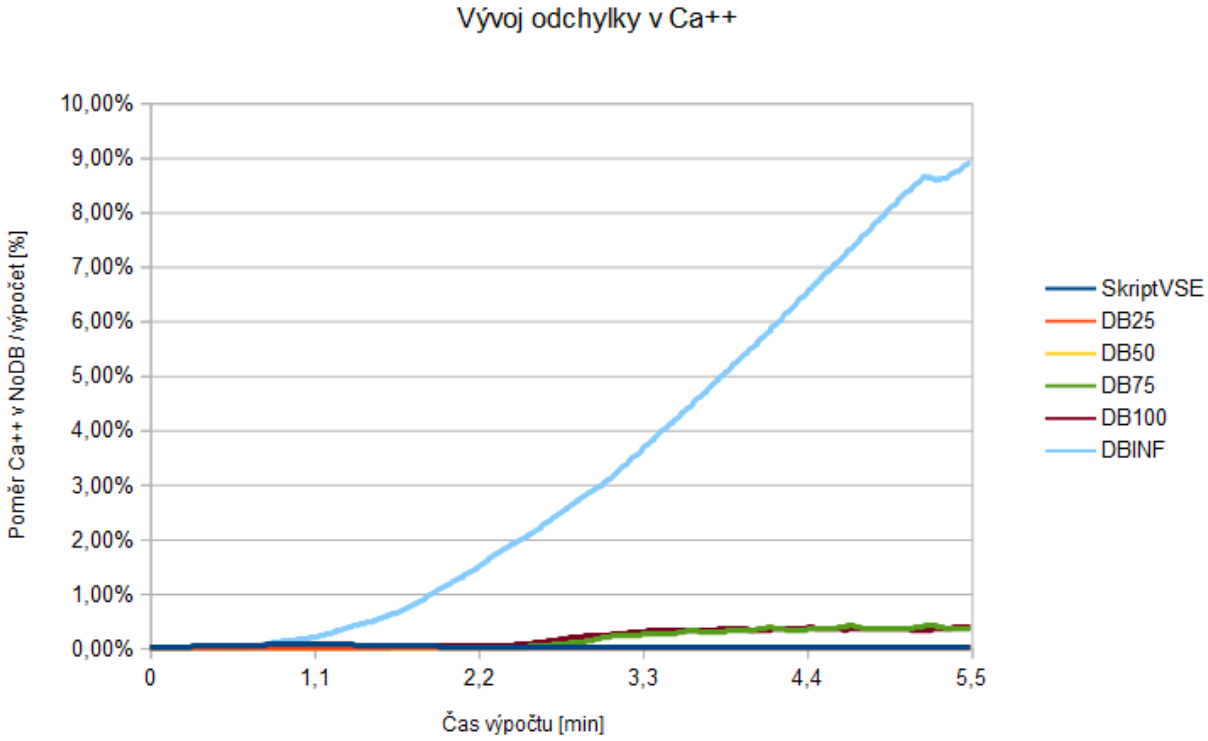
Obrázek 12: Graf vývoje počtu čtení z databáze – 50 buněk 1% tolerance



Obrázek 13: Graf vývoje časové náročnosti – 50 buněk 1% tolerance

Jak je však možné pozorovat na obrázku 14, kde je uveden graf vývoje odchylky v Ca^{++} všech výpočtů od výpočtu NoDB, velké urychlení v tomto případě také znamenalo velkou odchylku ve výsledcích. Výpočet DBINF, který dokázal časovou náročnost snížit ze dvou hodin na tři minuty, vykazoval na druhou stranu velké, téměř až 9% odchylky.

Ostatní druhy výpočtu, které používají omezenou databázi, vykazují odchylky velmi malé (platí pro výpočty DB100 a DB75) nebo úplně zanedbatelné (platí pro výpočty DB50 a DB25).



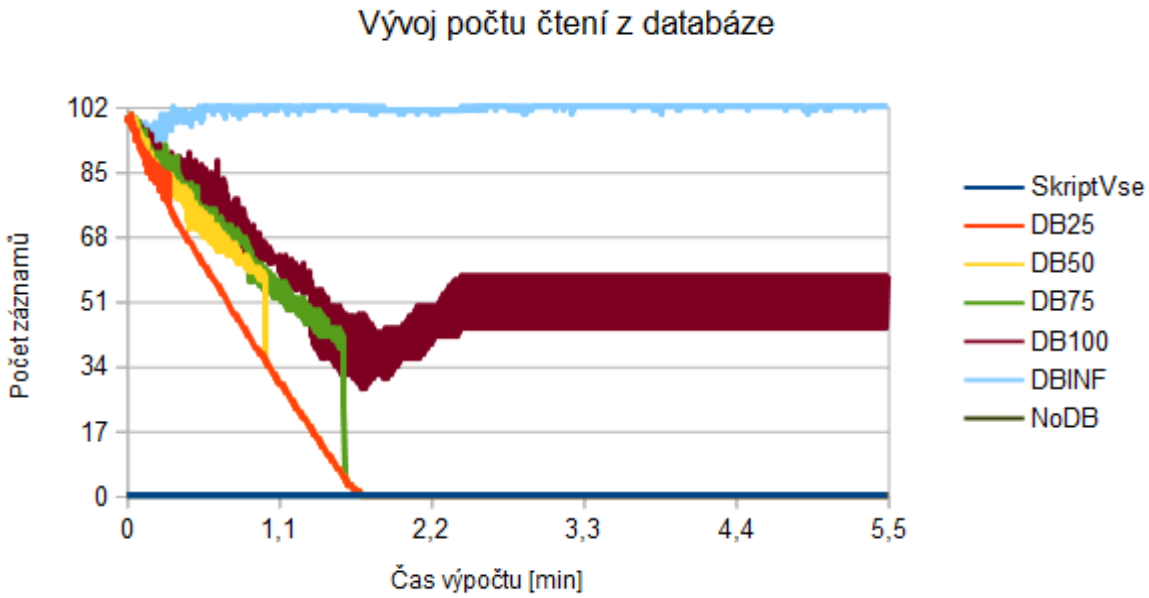
Obrázek 14: Graf vývoje odchylky v Ca++ od výpočtu NoDB – 50 buněk 1% tolerance

3.4 Testování modelu 100 buněk 1% tolerance

V této kapitole jsou zvoleny parametry počet buněk na 100, odpovídající časový krok je 0,005 minuty, a je zvolena 1% tolerance při vyhledávání v databázi. V textu práce budou nyní, na rozdíl od kapitoly 3.2, uvedeny jen některé grafy a analýzy. Kompletní seznam grafů je možné nalézt na přiloženém CD.

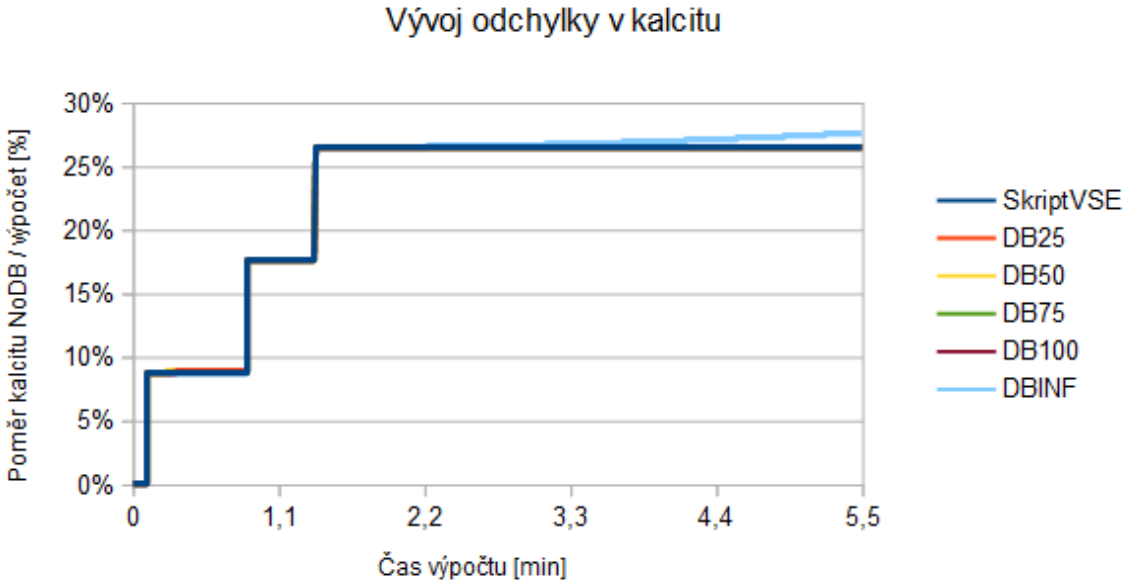
Na grafu, který je vidět na obrázku 15, je vyobrazen vývoj počtu čtení výsledků z databáze v průběhu výpočtu. Je zde možné si všimnout podobnosti s odpovídajícím grafem v kapitole 3.3, kde byla zkoumána kolona s padesáti buňkami. Tato podobnost je vidět na výpočtu DB25, který má totožný průběh v případě kolony o sto buňkách stejně jako v případě kolony o padesáti buňkách. Navíc se i u výpočtů s omezením databáze na 50 a 75 záznamů projevilo, že jsou příliš

omezené. Dá se tedy předpokládat, že velikost databáze musí být zvolena v závislosti na počtu buněk kolony tak, aby byl výpočet efektivní. Více bude tato závislost probrána v kapitole 3.5.



Obrázek 15: Graf vývoje počtu čtení z databáze – 100 buněk 1% tolerance

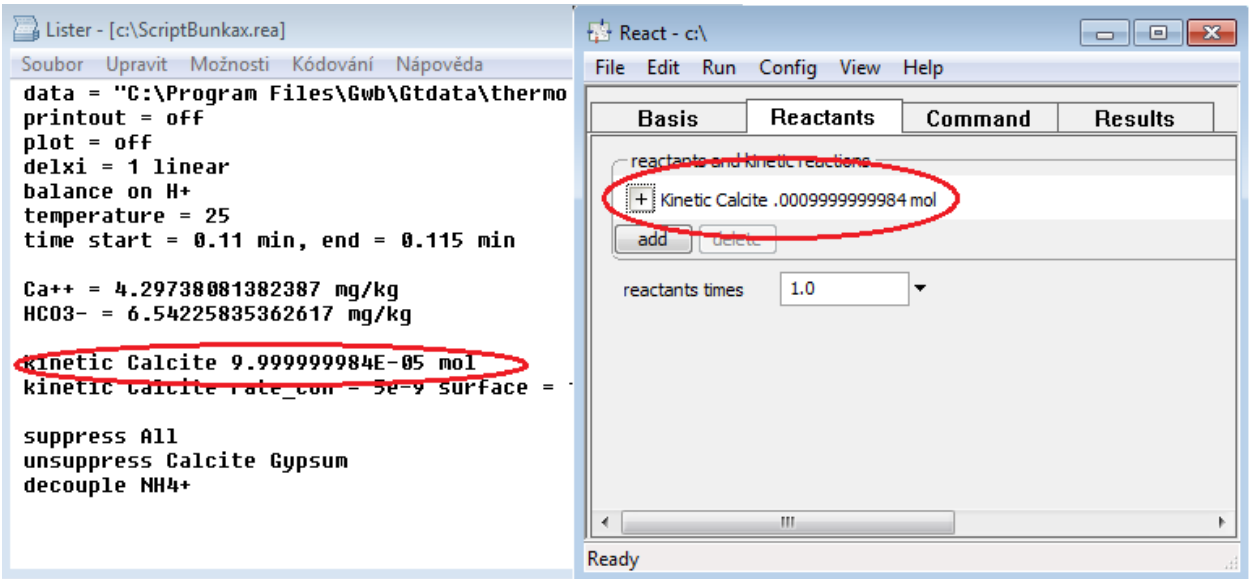
Nečekaný výsledek však přinesla analýza odchylek výpočtů od výpočtu NoDB, který nepoužívá ke svým výpočtům databázi. Na grafu 16 je vyobrazen vývoj odchylky v kalcitu pro výpočty vzhledem k výpočtu NoDB. Na grafu je možné vidět veliké skoky v odchylkách, které doposud nebyly nikde pozorovány. Na těchto skocích je zajímavé, že všechny výpočty mají vzájemně velmi podobné odchylky v kalcitu, ovšem všechny se liší od výpočtu NoDB.



Obrázek 16: Graf vývoje odchylky kalcitu - 100 buněk 1% tolerance

Tento výpočet byl tedy podroben bližšímu zkoumání, protože se zjevně jedná o chybu programu. Žádný doposud provedený výpočet totiž takové skoky v odchylkách nevykazoval. Při podrobnější zkoumání bylo ovšem zjištěno, že se nejedná o chybu programu Transprot ani o chybu rozhraní Konnektor, ale o chybu programu React.

Tato chyba je zdokumentována na obrázku 17. Na tomto obrázku je v levé části vidět skript vygenerovaný rozhraním Konnektor. Na pravé části je vidět grafické rozhraní programu React po spuštění uvedeného skriptu. Zvýrazněn je chybně načtený údaj. Přestože ve skriptu je kalcitu přibližně 10^{-4} molu, program po svém spuštění ukazuje přibližně hodnotu 10^{-3} molu, což je chybný údaj. Byla také snaha zjistit, za jakých okolností se toto chybné načítání děje a za jakých neděje, nebyla ovšem objevena žádná souvislost. Informace o této chybě bude odeslána tvůrcům programu React.



Obrázek 17: Ukázka chyby objevené v programu React

3.5 Shrnutí dosavadních poznatků z testování

Na základě doposud provedených testů je možné shrnout poznatky, které je možné brát v úvahu při provádění dalších testů. Shrnutí bude provedeno na základě srovnání přechozích tří kapitol.

Prvním parametrem, který byl sledován, bylo využití databáze. Tabulka 1 ukazuje srovnání využití databáze výpočtů typu DBINF, tedy těch, které vůbec nevyužívají promazávání databáze. V tabulce je uveden počet záznamů, který byl v databázi na konci výpočtu, a četnost čtení z databáze.

Tabulka 1: Tabulka využití databáze výpočtů typu DBINF

Počet buněk	10	50	100
Počet záznamů v databázi	352	610	698
Počet výpočtů reakční části	1320	28600	112200
Četnost čtení z databáze	73,33%	97,87%	99,38%

V tabulce je možné vidět, že přestože na koloně, která obsahovala sto buněk, bylo provedeno téměř čtyřikrát více výpočtů reační části než na koloně, která obsahoval padesát buněk, počet záznamů v databázi se změnil jen minimálně. Dále je možné pozorovat, že s přibývajícím počtem buněk kolony roste i četnost čtení výsledků z databáze a jejich aproximace. Je tedy možné usuzovat, že pro kolony s velkým počtem buněk bude více výpočtů aproximováno a může docházet ke množení chyby a ovlivnění celkového výstupu. Z toho vyplývá, že je vhodné počet záznamů v databázi omezit shora.

Z grafů 12 a 15 je možné dále vyčíst, že pokud je databáze příliš omezená, dojde v určitý časový okamžik k tomu, že přestane být úplně využívána a celý algoritmus ztrácí smysl. Z toho vyplývá i omezení maximálního počtu záznamů v databázi zdola. Vzhledem k těmto dvěma grafům je možné usuzovat, že aby byl výpočet efektivní, nemůže být databáze omezena na méně záznamů, než je počet buněk v dané koloně.

Druhým parametrem, který byl sledován, byla časová náročnost výpočtů. Ze srovnání grafů 7, 8 a 12, 13 je možné usuzovat, že vývoj časové náročnosti je závislý na tom, kolik záznamů je přečteno z databáze a kolik počítáno programem React. Dále je možné si všimnout, že původní způsob časové optimalizace, SkriptVSE, má srovnatelnou časovou náročnost, jako výpočty, které používají vhodně omezenou databázi, tak jak bylo popsáno v předchozích odstavcích.

Třetím sledovaným parametrem byla velikost odchylek různých výpočtů od výpočtu NoDB, který používá pro výpočet reakční části pouze volání programu React. Na grafech 10 a 14 je možné pozorovat, že pokud výpočet používal neomezenou databázi, měly tyto odchylky rostoucí charakter v průběhu výpočtu. Naopak u výpočtů, které používají databázi omezenou, je možné pozorovat, že růst odchylky v průběhu výpočtu je podstatně menší.

3.6 Srovnání modelu 50 buněk, různé tolerance

Jako referenční výpočet ke srovnání vlivu použité tolerance byl vybrán výpočet, který využívá kolonu o padesáti buňkách. Je tomu tak z toho důvodu, že se ukázalo, že deset buněk je

příliš málo na to, aby se při tomto srovnání dalo něco užitečného vypozorovat a kolony o sto buňkách nepřinesly žádné přelomové informace oproti kolonám o padesáti buňkách.

V této kapitole budou nejprve porovnány nejrychlejší výpočty, tedy výpočty, které používají neomezenou databázi. V tabulce 2 je uvedeno srovnání velikosti databáze, doby výpočtu a maximálních dosažených průměrných odchylek za jeden časový krok pro výpočtu typu DBINF v závislosti na použité toleranci.

Tabulka 2: Srovnání tolerancí pro výpočet DBINF

Tolerance	1,0%	0,5%	0,1%
Velikost databáze	610	1071	4058
Doba výpočtu [min:s]	03:04	05:30	20:23
Max odchylka pH [%]	0,46%	0,42%	0,31%
Max odchylka Ca^{++} [%]	8,90%	8,15%	6,16%
Max odchylka kalcit [%]	0,82%	0,76%	0,61%

Z tabulky je možné vidět, že při použití přesnějšího výpočtu roste poměrně rychle časová náročnost výpočtu, ovšem vliv na maximální odchylku není tak zásadní. Pro použitý model je tedy možné usuzovat, že pokud je použit výpočet s neomezenou databází, nemá smysl používat menší toleranci než 1% a to z důvodu velkého nárůstu časové náročnosti pro malý pokles v maximálních odchylkách.

Jako další budou stejným způsobem, který byl popsán v předchozích odstavcích, srovnány výpočty typu DB75, tedy s databází omezenou na maximálních 75 záznamů. Tento typ byl vybrán z následujících důvodů. Databáze, které jsou omezeny na menší počet záznamů, nemá smysl uvažovat z toho důvodu, že jsou pod nebo příliš blízko spodní hranice, která byla popsána v kapitole 3.5.

V tabulce 3 je zobrazeno podobné srovnání, které bylo provedeno pro výpočet DBINF. Řádek velikost databáze, který v tomto srovnání nemá smysl uvažovat, byl nahrazen procentuální hodnotou, kolik výsledků reakční části bylo načteno z databáze.

Na rozdíl od výpočtu, který využíval neomezenou databázi, je možné si nyní všimnout, že nárůst časové náročnosti není tak rychlý s použitím různých tolerancí a úměrně se snižují i dosažené maximální odchylky. Na druhou stranu je nutné poznamenat, že i při použití jednoprocentní tolerance jsou všechny odchylky menší než 0,5% a mohou být označeny za

dostačující.

Tabulka 3: Srovnání tolerancí pro výpočet DB75

Tolerance	1,0%	0,5%	0,1%
Poměr čtení z databáze [%]	78,82%	74,25%	55,92%
Doba výpočtu [min:s]	28:24	36:41	62:16
Max odchylka pH [%]	0,07%	0,05%	0,02%
Max odchylka Ca^{++} [%]	0,42%	0,17%	0,02%
Max odchylka kalcit [%]	0,04%	0,02%	0,004%

Z provedených analýz je tedy možné usuzovat, že při použití vhodně omezené databáze jsou výpočty s 1% tolerancí dostatečně efektivní. Pro výše zmiňovaný případ, tedy kolonu o 50 buňkách a toleranci 1%, je také možné pozorovat, že použití výpočtu typu DB75 vykazuje srovnatelné výsledky s původním typem optimalizace, tedy použití skriptu SkriptVSE. Z toho důvodu je v současné verzi rozhraní Konnektor implementována 1% tolerance.

Závěr

Práce se zabývá návrhem, implementací a testováním rozšíření programu Transport, který modeluje kolonové experimenty. Toto rozšíření bylo vyvinuto za účelem snížení časové náročnosti programu.

Během práce byl vytvořen algoritmus, který rozhoduje, zda výsledek reakční části výpočtu bude počítán externím programem React nebo zda bude načten z databáze. Algoritmus se rozhoduje pro každou buňku kolony, pro kterou má být počítána reakční část výpočtu, následovně. Nejprve se prohledá databáze, zda již není v databázi uložen výsledek reakčního výpočtu, který byl počítán se stejným nebo dostatečně podobným zadáním. V případě, že je takový výsledek nalezen, je algoritmem aproximován výsledek podle načteného záznamu, aby byla snížena vzniklá odchylka od případu, kdy by byl výsledek počítán programem React. V případě, že v databázi nebyl nalezen žádný vhodný výsledek, je pro výpočet reakční části volán externí program React a výsledek, který je takto získán je uložen do databáze. Tím dochází k zabránění opakovanému volání programu React se stále stejnými vstupy a ke zrychlení celého běhu programu. Tento algoritmus byl implementován do programu Transport.

Během práce bylo také testováno, za jakých podmínek je tento algoritmus efektivní. Jako parametry efektivity byla brána doba trvání výpočtu a velikost vzniklých odchylek v porovnání s výpočty, které nevyužívají databázi. Při testování byly výpočty spouštěny na kolonách s různým počtem buněk, s různým maximálním počtem záznamů v databázi a s různou použitou tolerancí. Celkem tam vzniklo několik kombinací těchto tří parametrů, jejichž výsledky byly vzájemně porovnány.

Bylo zjištěno, že rozšíření je z hlediska časové náročnosti efektivní zejména pro kolony s velkým počtem buněk. Dále bylo zjištěno, že aby byl výpočet efektivní, musí mít použitá databáze omezený maximální počet záznamů. Pokud je však tento maximální počet záznamů v databázi menší než počet buněk kolony, dochází opět k poklesu efektivity. Dále bylo zjištěno, že změnou tolerance, která je použita při vyhledání v databázi, se poměrně zásadně mění časová náročnost, ovšem změna v odchylkách od výpočtu pomocí programu React nemusí být tak značná.

Program Transport již před touto prací obsahoval jeden typ časové optimalizace. Program byl upraven tak, aby podporoval původní typ optimalizace i tento nově vzniklý. Při implementaci

byl také brán ohled na to, že na programu Transport mohou být implementovány další typy časové optimalizace například paralelizace, se kterou se do budoucna také počítá a kterou je možné na tuto práci navázat. Proto byl program upraven tak, aby přidání dalšího typu případně rozšíření tohoto nově vzniklého typu bylo co nejjednodušší.

V práci je porovnáván nově vzniklý algoritmus zejména s výpočty, který nevyužívají žádný typ časové optimalizace. Na tuto práci je tedy možné navázat porovnáním těchto dvou algoritmů a určením podmínek, za kterých je jeden nebo druhý typ optimalizace výhodnější.

Seznam použité literatury

[1] ŽABKA, Vratislav. 2009, *Simulace chemických reakcí v kolonovém experimentu* [online]. Diplomová práce. Technická univerzita v Liberci. Fakulta mechatroniky, informatiky a mezioborových studií. Dostupné z: http://artec.tul.cz/?content=upload/Diplomov%C3%A1%20pr%C3%A1ce.pdf&lang=cs

[2] ŽABKA, Vratislav, *Program Transport v2.2*, dokumentace. Liberec 2011. Technická univerzita v Liberci

[3] BETHKE, Craig M. a YEAKEL, Sharon, *The Geochemist's Workbench ®, Releace 9.0: Reaction Modelig Guide* [online]. University of Illnois, Urbana, 2013. [citováno 5. 4. 2016] Dostupné z: https://www.gwb.com/pdf/GWB9/GWBrxnmodeling.pdf

[4] BARTHOLOMEW, Daniel. SQL vs. NoSQL. *Linux Journal* [online]. 2010, 2010(195), 54 - 59 [cit. 2016-04-06]. Dostupné z: http://www.linuxjournal.com/article/10770

[5] PITTER, Pavel. Výpočet celkové mineralizace a její význam v hydrochemii. *Chemické listy*. 1998, **92**(10), 772 - 776. ISSN 1213-7103.

Příloha A: Obsah přiloženého CD

- Text diplomové práce
  - StrajtMichalDiplomovaPrace.pdf
  - StrajtMichalDiplomovaPrace.odt
- Zdrojový kód programu
- Spustitelný soubor programu s připravenou úlohou
- Diagramy tříd programu
- Kompletní výsledky testování
  - Výsledky pro 1% toleranci
  - Výsledky pro 0,5% toleranci
  - Výsledky pro 0,1% toleranci
  - Srovnání výsledků 50 buněk, různé tolerance