TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Paralelizace nástroje pro počítačovou podporu analýzy citlivosti modelů

Diplomová práce

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

Autor práce: **Bc. David Müller**
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. David Müller**
Osobní číslo:	**M14000171**
Studijní program:	**N2612 Elektrotechnika a informatika**
Studijní obor:	**Informační technologie**
Název tématu:	**Paralelizace nástroje pro počítačovou podporu analýzy citlivosti modelů**
Zadávající katedra:	**Ústav mechatroniky a technické informatiky**

Zásady pro vypracování:

1. Na základě poznatků získaných při řešení semestrální práce navrhněte způsob využití několika procesorů/jader pro paralelní výpočet několika úloh simulačním softwarem (např. FEFLOW) pro účely zpracování analýzy citlivosti.
2. Návrh implementujte.
3. Otestujte funkčnost a škálovatelnost 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] **D. Müller (2015), Rozšíření nástroje pro počítačovou podporu analýzy citlivosti hydrogeologických modelů, semestrální projekt FM TUL**

[2] **P. Tvrdík, Parallel algorithms and computing. Praha: ČVUT, 2009. ISBN 80-01-02267-6.**

[3] **FEFLOW 6.1 - User Manual, http://www.feflow.info/uploads/media/users_manual.pdf**

Vedoucí diplomové práce:	**doc. Ing. Jan Šembera, Ph.D.** Ústav mechatroniky a technické informatiky
Konzultant diplomové práce:	**Ing. Jiří Hnídek, Ph.D.** Ústav nových technologií a aplikované informatiky
Datum zadání diplomové práce:	**10. října 2015**
Termín odevzdání diplomové práce:	**16. května 2016**

prof. Ing. Václav Kopecký, CSc.
děkan

Technická univerzita v Liberci
L.S.
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: 16.5.2016

Podpis: 

Poděkování

Rád bych tímto poděkoval vedoucímu své diplomové práce doc. Ing. Janu Šemberovi, PhD., za odborné vedení a cenné rady při zpracovávání a psaní této diplomové práce. Dále bych chtěl poděkovat svým rodičům a sestře za podporu nejen při psaní této práce ale i během mého studia.

Abstrakt

Diplomová práce se zabývá paralelizací nástroje pro počítačovou podporu analýzy citlivosti hydrogeologických modelů. Cílem je paralelizovat výpočty softwaru, který dokáže analyzovat hydrogeologické modely a následně vyhodnotit jejich citlivost na změnu vstupních parametrů. K výpočtům dějů, které probíhají uvnitř modelů jako například proudění podzemní vody a transport hornin, je použit software FEFLOW. Výpočty probíhají na výpočetním clusteru a o interpretaci výsledku se stará aplikace na lokálním počítači. Jako paralelizační nástroj pro jazyk Java byla zvolena knihovna MPJ Express.

Dokument je rozdělen do tří hlavních částí. První z nich je část teoretická. Ta se zabývá vysvětlením pojmů důležitých pro pochopení práce a použitými prostředky či nástroji. Druhá část práce se věnuje samotnému vývoji programu, který je hlavním cílem práce. Poslední část práce se věnuje testování aplikace, testovacímu modelu, srovnání rychlostí výpočtu za použití různého počtu výpočetních uzlů a interpretací zjištěných výsledků.

Klíčová slova

FEFLOW, analýza citlivosti, vývoj programu, paralelizace, MPJ Express

Abstract

The thesis deals with parallelization of a tool for computer-aided analysis of sensitivity of hydrogeological models. The aim is to parallelize the calculations of the software that can analyze hydrogeological models and then evaluate their sensitivity to changes in input parameters. The calculations of the processes that take place inside models such as groundwater flow and transport of rocks are made with the help of the FEFLOW software. Calculations are performed in a computing cluster and the results are interpreted on a local computer. The MPJ Express library was chosen as the paralleling Java tool.

The document is divided into three main parts. The first one is theoretical and explains terms relevant for the understanding of the work and also describes the resources and tools that were used. The second part is dedicated to the development of the program, which is the main goal of the work. The last part is dedicated to application testing, test model, the comparison of calculation speed using a variable number of compute nodes and interpretation of the obtained results.

Keywords

FEFLOW, analysis of sensitivity, program development, parallelization, MPJ Express

Obsah

Úvod

Důvod, proč jsem si vybral dané téma, je prostý. Mohl jsem tak plynule navázat na svou bakalářskou práci, jejíž téma mě velmi zaujalo a řešení dané problematiky bavilo. Rozvoj znalostí a zkušeností vývoje softwaru mi připadá jako velmi dobrá investice k pozdějšímu uplatnění v praxi. Další pro mne velmi vítanou zkušeností byla možnost vyzkoušet si vývoj aplikace, která je určená pro paralelní výpočty a je spuštěna na výpočetním clusteru. Po studiu bych se chtěl programování věnovat a tak má pro mne vývoj takovéto aplikace velký přínos.

Cílem práce je paralelizovat nástroj pro počítačovou podporu analýzy citlivosti hydrogeologických modelů. Tento nástroj vznikl jako výsledek mé bakalářské práce. Nástroj analyzuje citlivost hydrogeologických modelů aplikace FEFLOW, což je software pro simulaci proudění podzemní vody, transportu rozpuštěných látek a transportu tepla porézním prostředím. Program, který je výsledkem této práce, bude schopen vypočítat stejné typy citlivostí, jako původní aplikace, ale jednotlivé výpočty budou spuštěny paralelně na výpočetním clusteru. Aplikace musí umět výsledky stáhnout zpět k uživateli, aby z nich mohl vyhodnotit, na změnu kterých vnějších či vnitřních vlivů je model nejcitlivější.

Existuje i řada dalších programů pro analýzu citlivosti hydrogeologických modelů. Jedním z nich je například UCODE_2005. UCODE_2005 je univerzální nástroj pro kalibraci modelů, užívaný např. pro kalibraci geochemických modelů, což je typický způsob využití analýzy citlivosti. Kalibrace je proces identifikace hodnot parametrů, při nichž výstupy modelu odpovídají nejlépe měřeným hodnotám pozorovaných veličin.

Práce se nezabývá vysvětlením problematiky hydrogeologických modelů ani odvozením rovnic, které se v ní vyskytují a to zejména kvůli jejímu rozsahu. Je předpokladem, že o nich má čtenář povědomí. Dalším omezením práce je školní licence pro FEFLOW. Ta má omezeny některé z funkcí. Tato licence je dále omezena na 25 počítačů. Z těchto důvodů nebude aplikace schopna využít více než 25 výpočetních uzlů počítačového clusteru.

Teoretická část práce se věnuje představení softwaru, který byl využit při jejím řešení, nebo byl pro řešení nezbytně nutný. Dále je v této části práce představen výpočetní cluster Hydra, který je využíván k paralelním výpočtům. Také jsou zde uvedené pojmy důležité pro pochopení analýzy citlivosti modelů a vysvětlené tři typy těchto citlivostí.

Praktická část práce se věnuje vývoji softwaru, který umožňuje počítat analýzu citlivosti paralelně. Tento program je hlavním přínosem této práce. Je zde uvedeno, jakým způsobem je rozšířená grafická část původní aplikace. Dále je zde vysvětlen způsob komunikace mezi aplikací a clusterem, technologie umožňující paralelní běh aplikace, komunikace mezi jednotlivými procesy a způsob předávání výsledků.

Závěr této práce je věnován testování výsledné aplikace. Je zde představen testovací model a jeho nastavení, včetně nastavení jednotlivých parametrů a pozorování. Tato část se také věnuje srovnání rychlosti výpočtu při použití různého počtu výpočetních uzlů. Poslední kapitola se věnuje interpretaci výsledků lokální a intervalové citlivosti.

1 Teoretický rozbor

Tato kapitola se zabývá teoretickou částí řešené problematiky. Obsahuje popis softwaru, který byl použit při řešení zadané úlohy, ať už se jedná o výpočet simulace proudění podzemní vody a transportu hornin nebo o tvorbu softwaru pro paralelní zpracování analýzy citlivosti. Dále jsou zde vysvětleny jednotlivé typy citlivostí hydrogeologických modelů a pojmy důležité pro jejich správné pochopení. Tato část práce obsahuje i popis výpočetního clusteru Hydra a základní informace o paralelizaci.

1.1 Software

V této části práce je rozebrán software pro simulaci proudění podzemní vody, transportu hornin a tepla porézním prostředím. Je zde představeno vývojové prostředí použité pro tvorbu výsledné aplikace. Jedna z podkapitol se zabývá programovacím jazykem, ve kterém je aplikace napsána. V další části kapitoly je představena grafická knihovna Swing. Poslední podkapitola obsahuje popis programu, který je touto prací rozšířen o paralelní zpracování analýzy citlivosti za pomoci výpočetního clusteru.

1.1.1 FEFLOW (v6.2)

FEFLOW (Finite Element Subsurface Flow System) je počítačový program pro simulaci proudění podzemní vody, transportu rozpuštěných látek a transportu tepla v porézním prostředí. Uživateli je umožněn výběr z několika typů proudění a to v prostředí saturovaném i nesaturovaném.

Tato práce využívá FEFLOW zejména pro vytváření hydrogeologických modelů, které jsou následně analyzovány a také pro spouštění výpočtů. FEFLOW generuje modely a jejich nastavení do textového souboru s příponou **.fem**. Tato nastavení se týkají okrajových podmínek, počátečních podmínek a tak dále. V souboru se také nacházejí veškerá nastavení nutná pro analýzu citlivosti konkrétního modelu. Jako poslední jsou zde výsledky výpočtu, které se do souboru zapíší po jejich ukončení.

1.1.2 Vývojové prostředí

Pro programování zadané úlohy se používá vývojové prostředí NetBeans IDE 8.0.2. Jedná se o svobodný software, který je distribuován zcela zdarma firmou Oracle Corporation. Primárně je určen pro objektově orientovaný programovací jazyk Java, ve kterém je také naprogramován. Dají se v něm však vytvářet i aplikace napsané v jazycích jako je PHP, C nebo C++. Součástí je také podpora pro webové aplikace s využitím HTML, CSS a JavaSriptu. Další informace o tomto prostředí jsou k nalezení na stránkách společnosti Oracle [1].

Základní verzi NetBeans lze rozšiřovat pomocí přídavných komunitních modulů. Těmito rozšířeními jsou například grafický debugger, syntaktická analýza zdrojového kódu a další nové funkcionality, které výrazně zjednodušují vývoj, ladění, testování programu a práci s NetBeans.

1.1.3 Java

Java je objektově orientovaný programovací jazyk vyvinutý společností Sun Microsystems. Oficiálně byl představen 23. května 1995. Jednou z jeho výhod je, že se jedná o multiplatformní jazyk a je tedy často využíván k vývoji programů, které mají běžet na různých platformách, jako jsou mobilní zařízení, desktopové počítače či distribuované systémy. Vývoji samotného jazyka se věnuje Patrick Naughton [2].

Jak už bylo zmíněno, Java je objektově orientovaný programovací jazyk. To znamená, že s výjimkou primitivních datových typů jsou všechny ostatní datové typy objekty. Java je interpretovaná a to má za následek, že místo skutečného strojového kódu se vytváří mezikód. Tomuto mezikódu rozumí interpret Javy a interpretuje jej. Z těchto důvodů je Java velmi kompatibilní, protože ji lze pustit všude tam, kde je nainstalován tzv. virtuální stroj Javy (Java Virtual Machine).

Syntaxe jazyka je jednoduchá, protože se v ní nevyskytují některé nízkoúrovňové konstrukce, jako jsou ukazatele, bezznaménkové číselné datové typy atd. Další vlastností je robustnost. Java je určena pro vývoj vysoce spolehlivých softwarů. Používá tzv. silnou typovou kontrolu, což znamená, že veškeré proměnné musí mít definovaný svůj datový typ. Další vlastností je generační správa paměti a použití tzv. Garbage collectoru. Progra-

mátor se nemusí starat o dealokaci paměti. Již zmíněný Garbage collector automaticky vyhledává nepoužívané části paměti a uvolňuje je pro další použití. Pro větší efektivitu je paměť rozdělena do více částí. Do konkrétních částí jsou objekty umisťovány podle délky jejich života a nad každou částí je spuštěn jiný algoritmus pro Garbage collection. Pro detailnější informace o Garbage collection viz článek společnosti Oracle [3].

1.1.4 Swing

Swing je knihovna grafických objektů pro platformu Java a navíc je nedílnou součástí Java SE od verze 1.2. Swing slouží k vývoji grafických aplikací, a proto zahrnuje běžné grafické prvky, kterými jsou okna, dialogy, tlačítka a další prvky. Ve vývojovém prostředí NetBeans lze tyto prvky snadno editovat a přidávat jim různé funkcionality.

1.1.5 Analyza_citlivosti

Analýza_citlivosti je počítačový program, který umožňuje za pomocí softwaru FEFLOW analyzovat citlivost pozorování hydrogeologických modelů na změnu vstupních parametrů. Pojmy parametr a pozorování vysvětluje kapitola *1.3 Analýza citlivosti*. Program dokáže analyzovat celkem tři druhy citlivostí a to zejména lokální citlivost, intervalovou citlivost a extrémní hodnoty. O jednotlivých citlivostech pojednává kapitola *1.4 Druhy citlivostí*. Uživatel má pak na výběr mezi různými typy pozorování.

1.2 Výpočetní cluster

Tato kapitola se zabývá problematikou výpočetních clusterů. Dále je zde představen výukový výpočetní cluster Hydra. V další části této kapitoly jsou vysvětleny základy paralelizace a dva různé přístupy k paralelizaci aplikací na výpočetním clusteru. Poslední podkapitola vysvětluje takzvané SSH.

1.2.1 Výpočetní cluster Hydra

Hydra je výukový výpočetní cluster, který je určen zejména k výuce paralelního programování a k testování paralelních aplikací. Jedná se o seskupení několika počítačů, které

spolu spolupracují a navenek se tváří, jako jedno velmi výkonné zařízení. Fakt, že je cluster složen z několika výkonných počítačů, přímo vybízí k nasazení paralelních aplikací. Složitá úloha je rozdělena na dílčí části, které jsou zpracovány na jednotlivých uzlech a výsledku je tak dosaženo mnohem dříve, než na klasickém stolním počítači. Z toho vyplývá, že úloha musí být k těmto účelům speciálně vyvinuta a tedy ne každou úlohu lze paralelizovat. Pro komunikaci mezi uzly se využívá např. MPI (Message-passing interface).

Hydra má jeden centrální uzel, který slouží k obsluze clusteru a neměly by na něm být spouštěny složité výpočty. Tímto uzlem je stroj Dell PowerEdge 1950. Výpočetním uzlům jsou pak úlohy přidělovány pomocí příkazu qsub, který je přiřadí do fronty a spustí, až bude mít dostatek volných prostředků. Těchto výpočetních uzlů je celkem 24. Jedenáct uzlů je Dell PowerEdge 1950 s dvěma dvoujádrovými procesory Xeon. Zbylých třináct uzlů je na počítačích Sun Fire V20z, které mají dva procesory Opteron s jedním jádrem. Celková kapacita clusteru tedy činí 48 procesorů se 70ti jádry. Propojovací sítí je Gigabit Ethernet 1000BASE-T. Operačním systémem tohoto clusteru je Linux CentOS 5.6. Další informace o tomto clusteru lze nalézt v článku na stránkách Hydra/techdetails [4].

1.2.2 Možnosti řešení paralelních částí programu

Příkaz QSUB

Příkaz qsub vytváří frontu jednotlivých úloh a ty jsou pak spouštěny a zpracovávány v závislosti na vytížení clusteru. Úlohy jsou ke zpracování předávány pomocí takzvaných dávkových souborů. Lze také určit, kolik výpočetních uzlů bude použito pro danou činnost nebo vyjmenovat přímo konkrétní uzly, které mají být využity.

Jednou z vlastností příkazu qsub je efektivní vytěžování clusteru. Je však potřeba myslet na fakt, že úlohy ve frontě zůstávají různě dlouhou dobu v závislosti na aktuálním stavu clusteru a také na množství uzlů, které úloha potřebuje ke svému spuštění. Jakmile jsou zdroje (tj. příslušné uzly nebo procesory) uvolněny, úloha je má k dispozici do jejího úplného skončení.

MPI

Message-passing interface (MPI) je standard pro paralelní aplikace a knihovny. MPI zahrnuje point-to-point předávání zpráv a globální operace. Zasílání zpráv probíhá na dvou úrovních. První z nich je zasílání zpráv mezi dvěma konkrétními procesy, které jsou identifikovány unikátním číslem. Druhá úroveň funguje na principu zasílání zpráv mezi skupinami procesů. Každý proces je mapován na jeden procesor a tím nemusí docházet k přepínání kontextů.

MPI je určeno především pro homogenní clustery a využívá model SPMD (Simple Program, Multiple Data). MPI je posazen do relační vrstvy referenčního modelu ISO/OSI, přičemž většina implementací používá ke komunikaci protokol TCP. Jedná se o API, které je nezávislé na programovacím jazyce. Další informace o MPI jsou k nalezení v knize s názvem A high-performance, portable implementation of the MPI message paging interface standard [5].

1.2.3 SSH

Zkratkou SSH se v informačních technologiích označuje tzv. Secure Shell. SSH má ale dva významy. Jedná se o zabezpečený komunikační protokol, který využívá pro spojení mezi počítači síťový komunikační protokol TCP/IP. Dále je touto zkratkou označován i shell[1] pro vzdálenou správu počítače, který je náhradou za zastaralý a nezabezpečený Telnet.

Bezpečná komunikace mezi klientem a serverem funguje na principu výměny klíčů. Server i klient mají dva typy klíčů. První klíč je privátní. Tento klíč je chráněn heslovou frází. Druhý z klíčů se nazývá veřejný. Veřejným klíčem serveru může uživatel snadno ověřit jeho totožnost. Veřejný a privátní klíč jsou vygenerovány vždy společně a jsou spolu úzce spjaty.

Na začátku komunikace si klient i server vymění základní údaje o komunikaci jako je například verze protokolu. Následně server pošle klientovi svůj veřejný klíč. Klient pak vytvoří klíč pro danou komunikaci a zašifruje ho svým privátním klíčem a i veřejným klíčem serveru a odešle ho. Protože data zašifrovaná privátním klíčem lze dešifrovat pouze

[1] Shell je textové uživatelské rozhraní, které je předchůdcem grafického.

příslušným veřejným klíčem, jsou ověřeni oba účastníci komunikace. Podrobné informace o protokolu SSH lze dohledat v dokumentu The Secure Shell (SSH) Protocol Architecture [6].

1.3 Analýza citlivosti

Analýza citlivosti se zabývá tím, jakým způsobem jsou výsledné hodnoty (pozorování) ovlivněny změnou vstupních parametrů. Příkladem problematiky, pro jejíž řešení je analýza citlivosti jedním z vhodných nástrojů, je hodnocení transportu látek v horninovém prostředí obklopujícím hlubinné úložiště radioaktivního odpadu. Lze tak určit vhodné prostředí, které oddálí průnik jaderného odpadu na povrch. Další možnosti využití a podrobnější informace o analýze citlivosti jsou popsány v bakalářské práci [7].

1.3.1 Parametry

Parametrem se rozumí konkrétní parametr modelu, který se zadává přímo ve FEFLOW a nějakým způsobem má vliv na hydrogeologický model. Jako příklad lze uvést pórovitost prostředí, piezometrickou výšku a další. Tyto parametry lze definovat v různých částech modelu. Některé jsou definovány pro konkrétní uzel, například okrajová podmínka piezometrické výšky, a jiné jsou zase definovány pro konkrétní element. Jako příklad lze uvést třeba propustnost.

1.3.2 Pozorování

Pozorováním se rozumí některá z výstupních hodnot, která je výsledkem výpočtu FEFLOW. Výsledný soubor obsahuje hodnoty různých fyzikálních veličin pro všechny časové kroky a pro danou citlivost je třeba zadat jednu z nich. Standardní typ pozorování umožňuje zkoumat situaci v konkrétním časovém kroku a konkrétním uzlu. Pro hledání maximálních a minimálních hodnot v různých časech a uzlech slouží pozorování zmíněná v následující kapitole.

1.3.3 Rozšířený typ pozorování

Aby bylo možné hledat minimální a maximální hodnoty v konkrétním čase nebo v konkrétním uzlu přes všechny časy, bylo potřeba zavést další rozšířené typy pozorování. Tyto typy pozorování jsou celkem tři. První z nich umožňuje hledat minimální či maximální hodnotu v konkrétním uzlu přes všechny časové kroky. Toto pozorování lze nastavit na zjišťování hodnoty minima (maxima) nebo času, kdy tato hodnota nastala. Druhý typ umožňuje hledat extrémní hodnotu v konkrétním čase ve všech zaznamenaných uzlech. Zde lze definovat, zda se má hledat hodnota extrému nebo uzel, ve kterém se tato hodnota nachází. Poslední pak hledá extrémní hodnotu přes všechny časy a uzly. Opět lze požadovat buďto hodnotu, čas nebo uzel, ve kterém extrém nastal.

1.3.4 Role počátečních podmínek

Při hledání minimálních či maximálních hodnot, které z modelu vystupují, je třeba brát v potaz počáteční podmínky. Některá pozorování mohou být počáteční podmínkou významně ovlivněna a to zejména u ustáleného proudění a transportu hornin. Zde totiž dochází v nultém časovém kroku k přemazání počátečních podmínek, protože pro další výpočet nemají význam. Je to z toho důvodu, že při ustáleném proudění se toto proudění v čase nemění a tedy počáteční podmínka zde postrádá na významu.

V takové situaci může program nalézt minimální/maximální hodnotu v počáteční podmínce a vracet tak zkreslené výsledky. V některých případech dojde k dělení nulou a program pak vrací výraz „NaN“ (Not a Number). V případě, že se jedná o neustálené proudění a transport rozpuštěných látek, ovlivňuje počáteční podmínka celý model, a tedy v takovém případě jsou výsledky v pořádku.

1.4 Druhy citlivostí

Tato kapitola a všechny její podkapitoly jsou převzaty z bakalářské práce [7]. Jsou zde uvedeny druhy citlivostí a jejich význam a možná interpretace. Dále kapitola obsahuje způsob výpočtu každé z citlivostí. V této části práce je také vysvětlen princip takzvaného škálování a důvody, proč jej dělat. Názvy parametrů v této kapitole nejsou překládány, aby bylo zachováno pojmosloví mezi touto prací a aplikací FEFLOW.

1.4.1 Lokální citlivost[2]

Lokální citlivost určuje, o kolik se změní výsledek pozorování v závislosti na změně jednoho z parametrů. Tato citlivost může být použita k odhadu vstupních parametrů, což se využívá především u kalibrace. Platí to však jen za předpokladu, že se hodnota pozorování mění vstupním parametrem pouze v malém rozsahu.

Lokální citlivost se vypočítá jako derivace výsledku pozorování, který vyplývá z modelu s konkrétními parametry. Pro výpočet je použit tento vzorec:

$$\frac{\partial y_i'(\vec{b})}{\partial b_j}, \tag{1.1}$$

kde y_i' odpovídá i-tému pozorování a b_j je j-tým parametrem. V reálném řešení je však nahrazena odhadem této derivace, který je definován vzorcem:

$$\frac{y_i'(\vec{b}+\Delta\vec{b})-y_i'(\vec{b})}{\Delta b_j} \tag{1.2}$$

Tento vzorec je možné představit si ve zjednodušené podobě jako:

$$\frac{poz(odchylka)\text{-}poz(původní\ hodnota)}{odchylka} \tag{1.3}$$

[2] Vzorce v této podkapitole (kromě vzorce (1.3) a (1.4)) byly převzaty z knihy Effective groundwater model calibration: with analysis of data, sensitivities, predictions, and uncertainty [8].

Poz(odchylka) odpovídá výsledné hodnotě v případě, že byl parametr navýšen o odchylku. *Poz(původní hodnota)* je výsledná hodnota v momentě, kdy byl parametr nastaven na původní hodnotu. Odchylka je rovna velikosti odchylky a *par(původní hodnota)* je původní hodnota parametru.

Bezrozměrná škálovaná citlivost

Při vyhodnocování citlivosti je obvykle užitečné porovnávat pouze relativní význam různých pozorování. Je totiž zřejmé, že jednotlivé citlivosti budou mít různý fyzikální rozměr v závislosti na vstupních datech. Jako příklad lze uvést situaci, kdy je jako parametr použit Hydraulic-head BC, který je dán v metrech a jako pozorování je nastaven Mass Concentration, který má rozměr [mg/l]. Výsledek citlivosti by pak měl rozměr [m·l/mg]. V případě, kdy tento případ nastane, je potřeba zavést tzv. škálování. To se využívá k vytvoření bezrozměrné citlivosti.

Princip škálování pro standardní pozorování spočívá v tom, že se výsledná citlivost, která vzniká při výpočtu vzorce (1.2), vynásobí původní hodnotou parametru a podělí výsledkem pozorování. Vzorec pro tento výpočet je znázorněn rovnicí (1.4).

$$\check{s}c_{ij} = \frac{y_i'\left(\vec{b} + \Delta\vec{b}\right) - y_i'(\vec{b})}{\Delta b_j} \cdot \left|\frac{b_j}{y_i}\right| \tag{1.4}$$

Složená škálovaná citlivost

V případě, kdy je potřeba zjistit, který parametr má největší vliv na všechna pozorování, bezrozměrná škálovaná citlivost nestačí. Je nutné zavést tzv. složenou škálovanou citlivost. Ta se vypočítá vzorcem (1.5).

$$s\check{s}c_j = \sqrt{\frac{\sum_{i=1}^{n}(\check{s}c_{ij})^2}{n}} \tag{1.5}$$

Jedná se o tzv. kvadratický průměr. Tímto průměrováním je zajištěno, že v případě, kdy má jeden parametr extrémní vliv na jedno pozorování a na ostatní vliv nemá, není tento fakt ve výsledcích potlačen. $s\check{s}c_j$ ve vzorci definuje složenou škálovanou citlivost pro j-té

pozorování. Suma, ve které se nachází $šc_{ij}$, vyjadřuje sumu všech dílčích citlivostí daného pozorování, to znamená citlivostí na všechny parametry a jmenovatel *n* je potom počet parametrů.

1.4.2 Intervalová citlivost

Intervalová citlivost je odvozena od citlivosti lokální, avšak zdaleka nevyjadřuje tutéž informaci. Na rozdíl od lokální citlivosti, která dává informaci o průběhu vývoje některého z pozorování, vyjadřuje intervalová citlivost extrémní hodnoty daného pozorování. To znamená, že díky této citlivosti jsme schopni určit rozsah hodnot, ve kterých se sledované pozorování nachází. Další rozdíl je ve velikosti odchylky a metodě škálování. Velikost odchylky je zde mnohonásobně větší. Je však potřeba ji volit obezřetně. V případě, kdy výsledky pozorování nejsou monotónní, dojde ke znehodnocení či zkreslení výsledku citlivosti. Je to dáno metodou výpočtu, kdy je pomocí derivace počítána tečna v bodě. Výsledný vzorec pro výpočet pak vypadá takto:

$$\frac{poz(max) - poz(min)}{par(max) - par(min)} \cdot \left| \frac{par(max) + par(min)}{poz(max) + poz(min)} \right| . \tag{1.6}$$

Poz(max) odpovídá výsledné hodnotě v případě, že byl parametr nastaven na maximum a *poz(min)* odpovídá výsledné hodnotě v případě, že byl parametr nastaven na minimum.

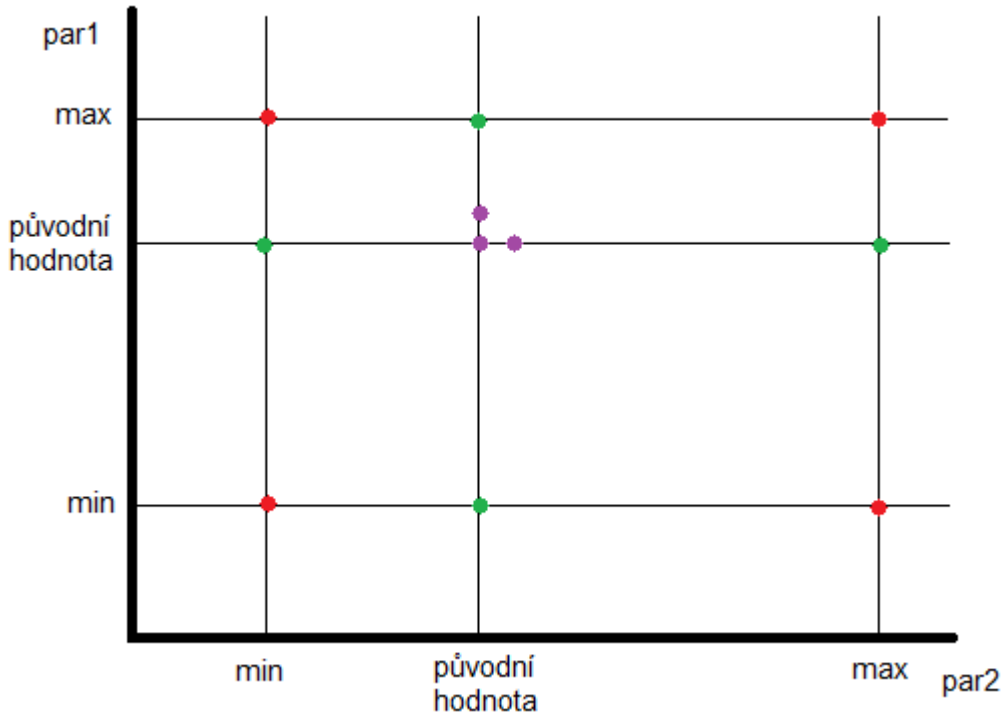
Metoda škálování standardního pozorování se, jak už bylo zmíněno, od metody používané u lokální citlivosti liší. Zde je výsledná citlivost vynásobena zlomkem, kde v čitateli je součet parametrů s maximální a minimální hodnotou a ve jmenovateli součet výsledků pozorování. Tento zlomek musí být v absolutní hodnotě, aby bylo zachováno znaménko vypočítané citlivosti.

1.4.3 Extrémní hodnoty

V tomto případě se nejedná o analýzu citlivosti v pravém slova smyslu. Tento výpočet umožňuje studovat chování modelu při změně všech vstupních parametrů. Pro výpočet jsou použity extrémní hodnoty všech parametrů, tedy jejich maximální a minimální hodnoty.

V případě, že jsou pro výpočet použity parametry dva, jsou na jeho začátku oba nastaveny na minimum. Po vypočítání výsledku je jeden z nich nastaven na maximum. V další části výpočtu se nastaví druhý z parametrů na maximum a první se vrátí na minimální hodnotu. Poslední výpočet pak proběhne s parametry nastavenými na jejich maxima.

Pro lepší představivost je zde uveden obrázek se znázorněním parametrů a jejich konkrétních hodnot pro výpočet jednotlivých citlivostí. Tento obrázek odpovídá modelu se dvěma parametry.



Obrázek 1 Definice parametrů

Osa x na *Obrázku 1 Definice parametrů* odpovídá hodnotám parametru pojmenovaným jako *par2* a osa y odpovídá hodnotám parametru *par1*. V grafu jsou znázorněny minimální, původní a maximální hodnoty obou parametrů. Pro výpočet intervalové citlivosti jsou využity hodnoty znázorněné zeleně. Pro výpočet citlivosti lokální jsou využity hodnoty znázorněné fialově a poslední hodnoty znázorněné v grafu červenou barvou jsou využity ke zjišťování extrémních hodnot.

1.4.4 Škálování rozšířeného typu pozorování[3]

Škálování lokální citlivosti pro standardní pozorování je popsáno v kapitole *1.4.1 Lokální citlivost*. Škálování lokální citlivosti pro pozorování typu *Min/Max v uzlu* probíhá stejným způsobem. I v případě pozorování typu *Min/Max v čase* se jedná o stejný výpočet, je-li výsledkem hodnota. Princip tohoto škálování je znázorněn vzorcem:

$$\frac{poz(odchylka) - poz(původní\ hodnota)}{odchylka} \cdot \left|\frac{par(původní\ hodnota)}{poz(původní\ hodnota)}\right| . \tag{1.7}$$

Poz(odchylka) odpovídá výsledné hodnotě v případě, že byl parametr navýšen o odchylku. Poz(původní hodnota) je výsledná hodnota v momentě, kdy byl parametr na původní hodnotě. Odchylka je rovna velikosti odchylky a par(původní hodnota) je původní hodnota parametru. Avšak v případě, kdy se uživatel ptá na uzel, ve kterém minimum popřípadě maximum nastalo, je způsob škálování odlišný. Místo rozdílu hodnot pozorování se zde používá vzdálenost uzlů a velikost úhlopříčky:

$$\frac{\|poz(odchylka) - poz(původní\ hodnota)\|}{odchylka} \cdot \left|\frac{par(původní\ hodnota)}{velikost\ úhlopříčky}\right| . \tag{1.8}$$

Výraz ||poz(odchylka)-poz(původní hodnota)|| odpovídá vzdálenosti uzlů, ve kterých se nalezená hodnota nachází. Tato vzdálenost má rozměr metry. Z těchto důvodů bylo potřeba pro škálování použít hodnotu, která má stejný rozměr a je vztažena ke konkrétnímu modelu. Ideální se zdá být velikost hrana jednoho elementu. Avšak ta se může lišit například zjemňováním sítě modelu. Z těchto důvodů je použita velikost úhlopříčky. Tato úhlopříčka je vedena mezi body, jež program nalezne v popisu uzlů daného modelu. Aplikace najde minimum pro x-ovou a y-ovou souřadnici a následně jejich maxima. Mezi těmito dvěma body je pak spočítána vzdálenost.

---

[3] Pozorování typu *Min/Max v uzlu* odpovídá situaci, kdy uživatel hledá minimální či maximální hodnotu v konkrétním uzlu přes všechny časové kroky. Pozorování typu *Min/Max v čase* umožňuje hledat extrémní hodnotu v konkrétním čase ve všech zaznamenaných uzlech. Pozorování *Min/Max v čase a uzlu* hledá extrémní hodnotu přes všechny časy a uzly.

U škálování lokální citlivosti pro pozorování typu Min/Max v čase a uzlu se používají oba již zmíněné vzorce. V momentě, kdy je potřeba škálovat hodnotu nebo čas, je použit vzorec (1.7). Na škálování polohy uzlu je použit vzorec (1.8).

Podobně to funguje u škálování intervalové citlivosti. Škálování intervalové citlivosti pro standardní pozorování je popsáno v kapitole *1.4.2 Intervalová citlivost*. Pro pozorování *Min/Max v uzlu, Min/Max v čase* a *Min/Max v čase a uzlu,* je-li výsledkem hodnota, se používá vzorec:

$$\frac{poz(max) - poz(min)}{par(max) - par(min)} \cdot \left| \frac{par(max) + par(min)}{poz(max) + poz(min)} \right|. \tag{1.9}$$

V případě, že výsledkem je doba minima, mají poz(min) respektive poz(max) ve vzorci (1.9) rozměr času. Jako poslední možnost je, že uživatele zajímá uzel, ve kterém hodnota nastala. V takové situaci se škáluje způsobem, kterým popisuje vzorec:

$$\frac{\|poz(max) - poz(min)\|}{par(max) - par(min)} \cdot \left| \frac{par(max) + par(min)}{velikost\ úhlopříčky} \right|. \tag{1.10}$$

2 Vývoj programu

Tato kapitola se zabývá samotným vývojem programu, který je výsledkem této práce. Vysvětluje, jakým způsobem byla aplikace rozdělena na serverovou a klientskou část. Dále je zde uveden program Master, který je umístěn na clusteru a poté vzdáleně spouštěn. Kapitola také vysvětluje, jakým způsobem spolu klientská část a serverová část aplikace komunikují. Jako poslední jsou pak uvedeny změny grafického rozhraní celé aplikace.

2.1 Rozdělení aplikace

Z důvodu zachování uživatelského rozhraní aplikace Analyza_citlivosti, byla aplikace rozdělena na dvě části. Klientskou část tvoří původní aplikace, jejíž ovládací prvky jsou rozšířeny. Jednotlivá rozšíření jsou uvedena v kapitole *2.4 Rozšíření grafické části*. Tyto nové ovládací prvky umožňují spouštět výpočet paralelně na výpočetním clusteru. Serverovou část tvoří nová aplikace Master, která je zodpovědná za provedení výpočtů pomocí aplikace FEFLOW, nalezení hodnot všech pozorování a uložení těchto hodnot do souboru. Klientská část si tento soubor stáhne ze serveru, provede analýzu citlivosti a pomocí tabulek zobrazí její výsledek uživateli. Důvody pro toto řešení jsou zejména zachování grafického uživatelského rozhraní a možnost spouštět analýzu citlivosti na lokálním počítači nebo na výpočetním clusteru v rámci jedné aplikace.

2.2 Program Master

Tato kapitola se zabývá vývojem programu. Je zde uveden způsob předávání vstupních parametrů. Dále je zde představen návrhový vzor pro vícevláknové aplikace, který aplikace využívá. Kapitola vysvětluje způsob komunikace mezi jednotlivými procesy a také jakým způsobem jsou počítány jednotlivé citlivosti.

2.2.1 Předání parametrů

Program Master je na clusteru spouštěn pomocí příkazové řádky. O způsobu komunikace mezi serverem a klientem pojednává kapitola *2.3 Komunikace mezi clusterem a aplikací*. Vlastní příkaz pro spuštění programu Master se skládá ze dvou částí. První část obsahuje spuštění Javy a nastavení MPI a druhá část obsahuje parametry, které jsou předány do aplikace pomocí proměnné *args*. Příklad příkazu, kterým lze aplikaci korektně spustit je následující:

```
mpjrun.sh -np pocetProcesoru -dev niodev -jar Master.jar pocetPozorovani
pocetParametru souborSModelem local interval extrem licence
```

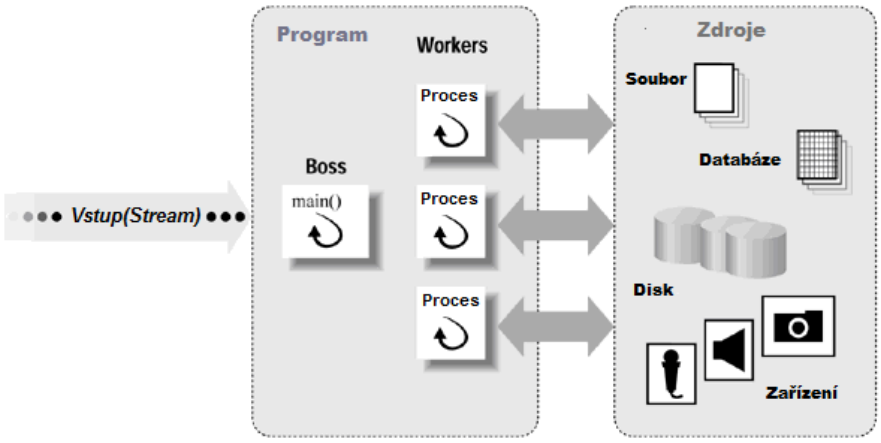
Tabulka 1 Vstupní parametry slouží k vysvětlení jednotlivých parametrů předchozího příkazu.

Tabulka 1 Vstupní parametry

Parametr	**Vysvětlení**
Mpjrun.sh	Slouží ke spouštění MPJ Express programů.
-np pocetProcesoru	Za pocetProcesoru je dosazeno číslo, které udává počet procesorů, které má mít aplikace k dispozici.
-dev niodev	Určuje zařízení, které má být využito ke komunikaci. V tomto případě je využito niodev (Java New I/O device). Niodev slouží ke spuštění MPJ Express programů na clusterech nebo počítačových sítích.
-jar Master.jar	Spuštěn má být soubor v JAR formátu s názvem Master.jar
pocetPozorovani	Číslo určující počet pozorování.
pocetParametru	Číslo určující počet parametrů.
souborSModelem	Řetězec obsahující cestu k souboru s modelem.
local	Počítat lokální citlivost Ano – 1, Ne – 0.
interval	Počítat intervalovou citlivost Ano – 1, Ne – 0.
extrem	Počítat extrémní hodnoty Ano – 1, Ne – 0.
licence	Řetězec obsahující IP adresu licenčního serveru pro FEFLOW.

2.2.2 Návrhový vzor Boss/Worker

Aplikace využívá návrhového vzoru Boss/Worker, který je detailně popsán společností IBM [9]. To znamená, že jedno vlákno (případně proces) funguje jako takzvaný boss (lze přeložit jako šéf), protože rozdává úkoly ostatním pracovním vláknům, kterým se říká workers (lze přeložit jako dělníci). Každý worker vykoná svou úlohu a následně informuje bosse, že je připraven přijmout další úkol. Na *Obrázku 2 Princip návrhového vzoru Boss/Worker* je nastíněn princip tohoto návrhového vzoru.



Obrázek 2 Princip návrhového vzoru Boss/Worker (Zdroj: [10], překlad z angličtiny)

V tomto případě probíhá komunikace mezi bossem a workerem pomocí MPI zpráv. O MPI pojednává kapitola *1.2.2 Možnosti řešení paralelních částí programu*. Boss na začátku zná počet svých workerů a počet úloh, které je potřeba vyřešit. Rozešle všem workerům zprávu s číslem výpočtu a čeká na odpověď. Číslo výpočtu určuje parametr, který má být změněn. Worker si načte soubor s modelem, změní příslušný parametr a takto upravený model uloží do nového souboru. Následně spustí aplikaci FEFLOW v konzolovém režimu, která provede výpočet pro model se změněným parametrem. Po skončení výpočtu zpracuje worker hodnoty požadovaných pozorování a zašle je zpět. Pak čeká na přidělení další úlohy. Je-li číslo úlohy hodnota *-1*, tak končí worker svou činnost, protože všechny úlohy již byly přiděleny ostatním workerům. Boss přijme takto zpracované hodnoty a uloží si je do paměti. Má-li k dispozici volnou úlohu, zašle její číslo zpět workerovi. Pokud jsou všechny úlohy zpracovány nebo přiděleny, zasílá boss hodnotu *-1*.

Po zpracování všech úloh uloží boss všechny výsledné hodnoty do souboru a ukončí svou činnost. Soubor s těmito výsledky má například následující strukturu:

```
/*Interval citlivost*/
Hodnota1    Hodnota2    Hodnota3      Hodnota4    Hodnota5
/**Interval citlivost**/
/*Lokalni citlivost*/
Hodnota1    Hodnota2    Hodnota3      Hodnota4    Hodnota5
/**Lokalni citlivost**/
/*Extremni hodnoty*/
Hodnota1    Hodnota2    Hodnota3      Hodnota4    Hodnota5
/**Extremni hodnoty**/
```

Tato struktura se mění jen v závislosti na citlivostech, které jsou skutečně počítány. Soubor je textový, aby byl pro uživatele snadno čitelný a aby ho mohl dále používat bez nutnosti složitých úprav či použití speciální programů.

2.2.3 Komunikace mezi procesy

Princip komunikace mezi procesy je vysvětlen v kapitole *2.2.2 Návrhový vzor Boss/Worker*. V této kapitole je vysvětlena technika, která se ke komunikaci využívá. Jedná se především o MPI popsaném v kapitole *1.2.2 Možnosti řešení paralelních částí programu*. MPI implementací pro Javu je několik. Pro účely vlastní práce byla vybrána implementace MPJ Express. MPJ Express byla vybrána proto, že je používána i pro školní účely. Jedná se především o předmět Distribuované programování. Na základě poznatků z toho předmětu bylo rozhodnuto, že je MPJ Express spolehlivé řešení a riziko vzniku problémů je minimální.

MPJ Express využívá techniku SPMD (Single program, Multiple Data). To znamená, že na všech používaných uzlech (počítačích v clusteru) je spuštěn stejný kód. Tento kód však pracuje s různými daty. Aby bylo možné rozdělit aplikaci na hlavní vlákno (boss) a vlákna pracovní (workers), je využit takzvaný rank. Rank je číslo identifikující konkrétní proces v komunikátoru. Komunikátor definuje skupinu procesů, které spolu mohou komunikovat. Komunikátor *MPI_COMM_WORLD* existuje vždy a obsahuje všechny procesy

v rámci aplikace. Tento komunikátor je v rámci práce využit jako jediný. Jak už bylo zmíněno, proces v komunikátoru je identifikován pomocí čísla rank. Rank s hodnotou nula má hlavní vlákno (boss) a ostatní čísla mají pracovní vlákna (workers). Rank čísla v rámci komunikátoru jsou vždy od nuly do čísla o jedno menší, než je počet procesorů zadaných při spouštění aplikace pomocí MPJ Express.

Na začátku metody *main(String[] args)* je potřeba zavolat inicializační metodu MPJ Express *MPI.Init(args)*. Dále je pomocí metody *MPI.COMM_WORLD.Rank()* zjištěn rank uzlu, na kterém je aplikace spuštěna. Na základě tohoto čísla se pak volají metody pro bosse nebo pro workera. Po dokončení celé aplikace je spuštěna metoda *MPI.Finalize()*.

Komunikace mezi procesy probíhá pomocí dvou metod. První z nich je metoda *MPI.COMM_WORLD.Send*. Tato metoda je blokující a to znamená, že se vykonávání kódu odesílatele v tomto místě zastaví, dokud jiný proces v rámci komunikátoru neodpoví metodou *MPI.COMM_WORLD.Recv*. V určitém případě by tedy mohlo dojít k uváznutí. Aplikace je však navržena tak, aby k uváznutí dojít nemohlo. Boss totiž vždy přijímá jakoukoliv příchozí zprávu a odpoví, kdežto worker přijímá pouze zprávy od bosse a na základě této zprávy rozhoduje, zda už nemá ukončit svou činnost. Kód zasílání a příjmu zpráv na straně bosse vypadá ve zjednodušené formě následovně:

```
/*Rozeslání instrukcí k prvnímu výpočtu všem procesům*/
for (zdroj = 1; zdroj < pocet_zdroju; zdroj++) {
      zprava = (String.valueOf(pocetVykonanych)).toCharArray();
      pocetVykonanych++;
      procesy.put(zdroj, zdroj);
      MPI.COMM_WORLD.Send(zprava, 0, zprava.length, MPI.CHAR, zdroj, znacka);
}
```

```
/*Čekání ve smyčce, dokud nejsou všechny výpočty hotové*/
while (true) {
            /*Příjem zprávy od libovolného procesu*/
            Status s = MPI.COMM_WORLD.Recv(mezivysledky, 0, pocetPozorovaniInt,
            MPI.DOUBLE, MPI.ANY_SOURCE, MPI.ANY_TAG);
            if (s.Get_count(MPI.DOUBLE) != 0) {
                System.arraycopy(mezivysledky, 0, vysledky[s.tag], 0,
                mezivysledky.length);
                if (pocetVykonanych >= pocetParametruInt) {
                    zprava = (String.valueOf(-1)).toCharArray();
                } else {
                    zprava = (String.valueOf(pocetVykonanych)).toCharArray();
                    pocetVykonanych++;
                }
                /*Odeslání odpovědi na příchozí zprávu*/
                MPI.COMM_WORLD.Send(zprava, 0, zprava.length, MPI.CHAR,
                s.source, tag);
            /*Odstranění již nepotřebných procesů ze seznamu zdrojů*/
            } else if (s.count == 0 && procesory.containsKey(s.source)) {
                procesy.remove(s.source);
            }
            /*Ukončení a zápis výsledků*/
            if (pocetVykonanych >= pocetParametruInt && procesory.isEmpty()) {
                HydraFileWriter.writeInterval(vysledky);
                break;
            }
}
```

V prvním cyklu rozešle boss všem workerům informaci o výpočtu, který mají provést. Každý worker je uložen do seznamu procesů. Následně ve smyčce čeká na zprávu s výsledky od jakéhokoliv z nich. Tento fakt je dán použitím *MPI.ANY_SOURCE* a *MPI.ANY_TAG* při volání metody *MPI.COMM_WORLD.Recv*. Po přijetí zprávy se kontroluje, zda není pole s výsledky prázdné. V takovém případě to znamená, že worker ukončil svou činnost a je odebrán ze seznamu zdrojů. Jestliže je pole s výsledky naplněno hodnotami, jsou tyto hodnoty uloženy do dvourozměrného pole, kde číslo řádku odpovídá číslu parametru, který byl v daném výpočtu změněn. Toto číslo je předáno pomocí takzvaného tagu zprávy. Po uložení výsledků pak boss opět volá metodu *MPI.COMM_WORLD.Send* s dalšími instrukcemi. V případě, že je seznam procesů prázd-

ný a počet vykonaných výpočtů je roven počtu parametrů, jsou všechny výpočty u konce a boss může pole s výsledky uložit do souboru. Na straně workera je potřeba zajistit příjem zpráv a odeslání správných odpovědí. Dále také spouštění výpočtu a zpracování výsledků. Kód zodpovědný za tuto činnost vypadá následovně:

```
char[] zprava;
double[] vysledky;
String s1;
while (true) {
    vysledky = new double[0];
    zprava = new char[60];
    Status s = MPI.COMM_WORLD.Recv(zprava, 0, 60, MPI.CHAR, 0, tag);
    int nrecv = s.Get_count(MPI.CHAR);
    s1 = new String(message);
    if ("-1".equalsIgnoreCase(s1.substring(0, nrecv))) {
       MPI.COMM_WORLD.Send(vysledky, 0, vysledky.length, MPI.DOUBLE, 0, tag);
       break;
    } else {
       IModelMaster model;
       Int cisloVypoctu = Integer.parseInt(s1.substring(0, nrecv));
       model = new ModelMasterExtrem(vstupniSoubor, myRank, licenceServer);
       vysledky = model.spust(cisloVypoctu);
       MPI.COMM_WORLD.Send(vysledky, 0, vysledky.length, MPI.DOUBLE, 0,
       cisloVypoctu);
     }
}
```

Jak je vidět, tento kód obsahuje pouze jeden cyklus. Na začátku je potřeba inicializovat vstupní buffer *zprava* pro příjem zprávy. Zpráva je přijata jako jednorozměrné pole znaků a to pouze od procesu, jehož rank je roven nule. Metoda *Get_count(MPI.CHAR)* třídy *Status* umožňuje zjistit, kolik znaků bylo skutečně odesláno. Tím lze ze vstupního bufferu vybrat pouze relevantní znaky. Pokud je zpráva *-1*, worker odešle bossovi prázdné pole typu *MPI.DOUBLE* a cyklus je ukončen. V opačném případě je vytvořen takzvaný model. Typ modelu závisí na aktuálně počítaném typu citlivosti. V tomto případě se jedná o citlivost s názvem extrémní hodnoty, o které pojednává kapitola *1.4.3 Extrémní hodnoty*. Metoda *spustit* třídy *ModelMasterExtrem* vrací jednorozměrné pole datového typu *double*. Worker toto pole následně odešle zpět bossovi. Aby mohl boss správně přiřadit výsledky ke konkrétnímu výpočtu, je číslo výpočtu předáno pomocí takzvaného tagu zprávy.

2.2.4 Výpočet lokální citlivosti

Pojem lokální citlivost byl vysvětlen v kapitole *1.4.1 Lokální citlivost*. Tato kapitola se zabývá pouze principem jejího výpočtu a zpracování. Průběh přidělování úloh jednotlivým pracovním vláknům/procesům[4] a následné zpracování výsledků je vysvětleno v kapitole *2.2.2 Návrhový vzor Boss/Worker*. Pracovní vlákno (worker) přijme číslo výpočtu, vytvoří si takzvaný model a poté ho spustí. Modelem se rozumí objekt typu *ModelMasterLokal,* který má metody pro výpočet a zpracování lokální citlivosti.

Třída *ModelMasterLokal* obsahuje metody pro čtení hydrogeologických modelů, načítání parametrů a pozorování, ukládání upravených modelů do souboru a spouštění výpočtu pomocí FEFLOW. Jediná veřejná metoda je však metoda *spustit(int cisloVypoctu).* Pro veškeré čtení a zápis do souborů je použito kódování UTF-8. V případě, kdyby nebylo kódování pevně nastavené, byly by některé znaky špatně čteny a to by zamezovalo správnému spouštění modelů ve FEFLOW. Tento neduh byl způsoben použitím třídy *PrintWriter* při zápisu do souboru a použitím třídy *BufferedReader* při čtení souboru, přičemž obě třídy používají standardně jiné kódování.

Pomocí konstruktoru je objektu *ModelMasterLokal* předán soubor s modelem, číslo procesu a IP adresa licenčního serveru FEFLOW. V konstruktoru je pak volána metoda *inicializaceModelu().* Ta obsahuje načtení modelu, parametrů a pozorování ze souboru, dále nastavení počátečních hodnot parametrů modelu a uložení upraveného modelu do dočasného souboru. Tento soubor je pak používán k analýze citlivosti a s původním souborem už se nikterak nepracuje.

Jak už bylo zmíněno, jediná veřejná metoda je metoda *spustit(int cisloVypoctu).* Argumentem této metody je číslo výpočtu. V případě lokální citlivosti je to tedy číslo parametru, který má být změněn. Na začátku je potřeba zjistit, zda už dočasný soubor s modelem existuje a případně jej vytvořit. Jelikož je model s počátečními hodnotami držen v paměti ve formě řetězce, stačí tento řetězec zkopírovat do řetězce nového a tomuto řetězci změnit příslušný parametr. Ke změně parametrů slouží třída ZmenParametr, která má statickou metodu *zmenParametr(String textFem, Parametr parametrZmen, int typParametru).*

[4] O vláknu lze hovořit v případě, kdy je MPI spuštěno v takzvaném multivláknovém režimu. V režimu clusteru se pak jedná o proces.

Argument *textFem* je řetězec s modelem, *parametrZmen* je parametr, který má být změněn a *typParametru* určuje, zda se má jako hodnota parametru použít minimální/maximální hodnota, původní hodnota či původní hodnota změněná o odchylku. Takto změněný model je uložen a použit k výpočtu.

Pro výpočty pomocí FEFLOW slouží třída *CentOSFeflow*. Tomu je v konstruktoru předána IP adresa licenčního serveru FEFLOW. Tato třída obsahuje jedinou metodu. Touto metodou je *runFeflow(String souborSVysledky, String souborProVypocet).* Argument *souborSVysledky* obsahuje cestu k souboru, který je výsledkem výpočtu aplikace FEFLOW. Argument *souborProVypocet* obsahuje cestu k souboru s modelem, který má být analyzován. Aplikace Feflow je spuštěna v konzolovém režimu jako samostatný proces. Příkaz pro spuštění Feflow vypadá následovně:

```
feflow62c -ascii -run -license server="IP_adresa" -dac souborSVysledky
souborProVypocet
```

V příkazu je za *IP_adresa* dosazena konkrétní IP adresa licenčního serveru a za *souborSVysledky* a *souborProVypocet* jsou dosazeny cesty k příslušným souborům. Po skončení výpočtu vytvoří FEFLOW soubor s výsledky, který může být následně podroben analýze.

Pro nalezení a uložení hodnot jednotlivých pozorování pro příslušný parametr slouží třída *Vysledky.* Tato třída obsahuje metodu *vytvorVysledky(ArrayList<IPozorovani> pozorovani, String cestaKVysledkum).* První z argumentů této metody je seznam pozorování. Druhým z argumentů je řetězec obsahující cestu k souboru s výsledky. Soubor s výsledky obsahuje kompletní popis hydrogeologického modelu a také výsledky pro veškeré zaznamenané časové kroky. Ve výsledcích se nachází hodnoty piezometrické výšky, Darcyho rychlosti v ose x, Darcyho rychlosti v ose y a koncentrace. Tyto hodnoty jsou zaznamenávány pro všechny uzly modelu.

Popis modelu je ukončen klíčovým slovem $END. Po něm následují hodnoty pro automaticky generované časové kroky nebo kroky pevně definované. Metoda postupně čte soubor s výsledky. Pokud je čas pozorování shodný s některým z časových kroků ze souboru, jsou uloženy tyto hodnoty.

Jestliže se takový časový krok v souboru nenachází, počítá se lineární kombinace dvou nejbližších kroků. Návratovou hodnotou metody *vytvorVysledky* je jednorozměrné pole datového typu double. Toto pole je odesláno MPI zprávou hlavnímu vláknu.

Hlavní vlákno neboli boss toto pole přijme a pomocí takzvaného tagu zprávy určí číslo parametru, kterému výsledky patří. Podle tohoto čísla si ho uloží do svého dvourozměrného pole s výsledky a pokračuje dál v činnosti. V případě, kdy má uloženy všechny výsledky, uloží boss tyto výsledky do textového souboru a program Master pokračuje výpočtem dalších druhů citlivostí, nebo končí svou činnost. Textový soubor byl zvolen úmyslně a to zejména proto, aby s těmito hodnotami měl uživatel možnost snadno pracovat a dále je používat.

2.2.5 Výpočet intervalové citlivosti

Pojem intervalová citlivost byl vysvětlen v kapitole *1.4.2 Intervalová citlivost*. Zde bude vysvětlena technika jejího výpočtu a zpracování. Princip komunikace mezi hlavním vláknem (boss) a výpočetními vlákny (workers) je vysvětlen v kapitole *2.2.2 Návrhový vzor Boss/Worker*. Boss na začátku rozešle čísla výpočtů zprávou MPI všem svým workerům. Číslo výpočtu udává číslo parametru ze seznamu parametrů, který má být pro danou analýzu změněn. Worker si vytvoří takzvaný model. V tomto případě se modelem rozumí objekt typu *ModelMasterInterval*. *ModelMasterInterval* má metody pro výpočet a zpracování intervalové citlivosti. Dále má metody pro čtení hydrogeologických modelů a jejich zpětné ukládání do dočasných souborů, metody pro načítání parametrů a pozorování a pro spuštění vlastního výpočetu pomocí FEFLOW. Pro čtení i zápis do souborů je použito kódování UTF-8, stejně jako tomu je u citlivosti lokální v předchozí kapitole. Poslední z metod slouží ke spojování dvou polí. V případě intervalové citlivosti se totiž za jedním výpočtem daného workera skrývají dva výpočty FEFLOW. V prvním výpočtu je na místo parametru dosazena jeho minimální hodnota. V druhém výpočtu je pak dosazeno maximum. Tyto hodnoty definuje uživatel při zadávání parametrů do aplikace.

Stejně jako v případě lokální citlivosti v kapitole *2.2.4 Výpočet lokální citlivosti* je objektu typu *ModelMasterInterval* pomocí konstruktoru předán soubor s původním modelem, číslo procesu a řetězec obsahující IP adresu licenčního serveru aplikace FEFLOW.

V konstuktoru je volána metoda *inicializaceModelu()*. Tato metoda obsahuje načtení hydrogeologického modelu, zvolených parametrů a pozorování ze souboru. Také nastavení počátečních hodnot všech parametrů a vytvoření dočasného souboru s nově upraveným modelem, který bude dále použit pro výpočty.

Jediná veřejná metoda je metoda *spustit(int cisloVypoctu)*. Argumentem této funkce je číslo výpočtu. Toto číslo udává číslo parametru ze seznamu parametrů, který má být v daném výpočtu změněn. Metoda *spustit* vrací pole typu double, ve kterém jsou uloženy všechny hodnoty pozorování pro daný parametr. Uvnitř této metody se spouští výpočet pro parametr nastavený v minimální hodnotě a následně pro parametr v hodnotě maximální. Po ukončení obou výpočtů jsou tato dvě pole spojena do jednoho a odeslána zpět bossovi. Jak konkrétně je spouštěn výpočet pomocí FEFLOW je vysvětleno v kapitole *2.2.4 Výpočet lokální citlivosti*.

2.2.6 Výpočet extrémních hodnot

Pojem extrémní hodnoty byl vysvětlen v kapitole *1.4.3 Extrémní hodnoty*. Tato kapitola se zabývá způsobem výpočtu tohoto typu citlivosti a jejím zpracováním. Princip komunikace mezi hlavním a pracovními vlákny je vysvětlen v kapitole *2.2.2 Návrhový vzor Boss/Worker*. Boss na začátku rozešle čísla výpočtů MPI zprávou všem svým workerům. Těchto výpočtu je dohromady 2^n, kde n je počet parametrů. Worker si vytvoří objekt typu *ModelMasterExtrem*. Při vytváření objektu se provede inicializace, tedy načtení parametrů, načtení pozorování, vytvoření dočasného souboru pro model se změněnými parametry a vytvoření spojového seznamu.

Tento seznam obsahuje řetězce složené pouze z jedniček a nul. Takových řetězců je v seznamu 2^n, kde n je počet parametrů. Počet znaků řetězce je roven počtu parametrů a žádný řetězec se v seznamu neopakuje. Podle čísla výpočtu worker určí, který z řetězců v seznamu má použít. Na základě znalosti tohoto řetězce jsou nastavovány parametry modelu. Parametru, na jehož pozici je v řetězci nula, je nastavena minimální hodnota. Pakliže se na jeho pozici nachází jednička, je nastaven na hodnotu maximální.

Stejně jako v případě ostatních citlivostí je jedinou veřejnou metodou třídy *ModelMasterExtrem* metoda *spustit(int cisloVypoctu)*. Metoda *spustit* vrací pole datového typu double,

ve kterém jsou uloženy všechny nalezené hodnoty pro jednotlivá pozorování. Toto pole je odesláno pomocí zprávy MPI zpět hlavnímu vláknu, které si jej uloží. Princip spouštění výpočtu pomocí FEFLOW je vysvětlen v kapitole *2.2.4 Výpočet lokální citlivosti*.

2.3 Komunikace mezi clusterem a aplikací

Tato část práce vysvětluje princip komunikace mezi clusterem a aplikací spuštěnou na lokálním počítači. Je zde vysvětlen návrhový vzor Observer. Dále se zde pojednává o šifrované komunikaci a knihovně JSCH, která se k této komunikaci využívá. V poslední podkapitole je pak uveden způsob uložení uživatelského nastavení, které se týká připojení ke clusteru.

2.3.1 Využití návrhového vzoru Observer

Protože grafická část aplikace běží v jednom vlákně, při spuštění komunikace s clusterem by přestala reagovat na podněty uživatele. Z tohoto důvodu implementuje třída zodpovědná za komunikaci (konkrétně třída *SSHDriver*) rozhraní *Runnable*. Na základě této skutečnosti je nutné implementovat metodu *run()*. Kód metody lze spustit v samostatném vlákně například tímto způsobem: *new Thread(new SSHDriver())*. Ideální způsob, jak zajistit komunikaci mezi grafickou částí a třídou *SSHDriver* je návrhový vzor Observer.

Ten zavádí možnost sledování změn mezi objekty. Jeden z objektů změní svůj stav a ostatní objekty na tuto změnu mohou reagovat, přičemž nedochází k přímé vazbě mezi sledovaným objektem a objekty ostatními. V Javě jsou nástroje pro použití tohoto návrhového vzoru již implementované. Další informace a možnosti využití návrhového vzoru Observer vysvětluje Kraval [11].

Třída *SSHDriver* tedy implementuje rozhraní *Observable*. Pokud dojde ke změně, o které je potřeba informovat GUI (grafické rozhraní aplikace), jsou volány metody *setChanged()* a *notifyObservers()*. První z metod říká, že v objektu došlo ke změně. Druhá z metod o těchto změnách informuje všechny přiřazené objekty.

Třída *Aplikace* je zodpovědná za vykreslování GUI a dědí od třídy *Observer*. Obsahuje tlačítko pro spuštění výpočtu na výpočetním clusteru. Při stisku tlačítka je objekt tohoto GUI přiřazen jako *Observer* objektu typu *SSHDriver*. Ten je spuštěn v samostatném vlákně. Dále je tlačítko nastaveno jako neaktivní do té doby, než bude výpočet ukončen. Třída *Aplikace* implementuje metodu *update(Observable o, Object o1)*. Ta se vykoná v případě, kdy je na sledovaném objektu zavolána metoda *notifyObservers()*. Metoda *update* nastaví tlačítko opět jako aktivní a dojde k vykreslení tabulek s výsledky.

2.3.2 Komunikace

Pro spojení s clusterem Hydra se používá SSH. V Javě ke komunikaci pomocí SSH slouží knihovna JSCH – Java Secure Channel. Jedná se o čistou implementaci SSH2. Knihovna je vydávána pod licencí BSD, což je licence pro svobodný software. Pro komunikaci aplikace na lokálním počítači s clusterem Hydra slouží třída *SSHDriver*, která právě výše zmíněnou knihovnu využívá. Další informace o JSCH jsou k nalezení na webových stránkách společnosti JCraft, Inc. [12].

Pro navázání spojení je zapotřebí znát uživatelské jméno, heslo a název serveru. Veřejný klíč serveru se pak stáhne po stisku tlačítka automaticky. Pro spojení pomocí JSCH je potřeba vytvořit objekt typu JSch. Dále je zapotřebí vytvořit tzv. *Session*. JSCH lze nastavit tak, aby se při spojení neověřoval veřejný klíč serveru. Při aktivaci této možnosti však uživatel přijde o všechny výhody zabezpečené šifrované komunikace. Proto aplikace veřejný klíč ověřuje. Při změně veřejného klíče je uživatel o této skutečnosti informován a je mu nabídnuta možnost klíč uložit. Tento klíč je uložen do souboru „knownHostsFile“, který vždy obsahuje pouze naposledy uložený klíč. Uživatelské nastavení, a to zejména jméno, heslo a název serveru, jsou uloženy v souboru „nastaveniHydry.txt“. Řešení navazování spojení pomocí jména uživatele a hesla bylo zvoleno z důvodu většího pohodlí uživatele. V tomto případě totiž uživatel nemusí vědět, jak generovat sadu klíčů, která by byla pro komunikaci využita.

Kód pro navázání spojení s Hydrou je následující:

```
JSch jsch = new JSch();
Session session = null;
/*Navázání spojení s Hydrou*/
try {
session = jsch.getSession(uzivatel, server, 22);
session.setPassword(heslo);
jsch.setKnownHosts(new FileInputStream(knownHostsFile));
session.setConfig("StrictHostKeyChecking", "yes");
session.connect();
}
```

Jak už bylo zmíněno, na začátku je vytvořen objekt typu *JSch*. Dále je vytvořena takzvaná *Session*. Ta je tvořena pomocí jména uživatele, také je zadán název serveru a číslo portu. Port číslo 22 je rezervován právě pro SSH spojení. Poté se *Session* nastaví heslo a známý veřejný klíč. Příkaz *setConfig* pak umožňuje rozhodnout, zda se veřejný klíč serveru má ověřovat či nikoliv. Po těchto nastaveních je možné navázat spojení se serverem.

SSHDriver slouží dále pro nahrávání a stahování souboru z Hydry. A také pro spouštění příkazu na clusteru. Pro všechny tyto úkoly se využívá již zmíněná knihovna JSCH. Pro nahrání souboru na Hydru je zapotřebí nejdříve vytvořit *Session* a následně komunikační kanál *ChannelSftp*. Pro nastavení cílového adresáře slouží metoda *cd(String cesta)*, která jako argument přijímá cestu k cílovému adresáři. Vlastní soubor je pak nahrán pomocí metody *put()*. Stahování souboru funguje velmi podobným způsobem, avšak místo metody *put()* je použita metoda *get()*. Metody pro tyto dva úkony jsou *sendToHydra()* a *loadFromHydra()*.

Kód metody *loadFromHydra()* vypadá následovně:

```
ChannelSftp channel;
try {
        channel = (ChannelSftp) session.openChannel("sftp");
        channel.connect();
        channel.cd(hydraPath);
        channel.get("results.txt", path);
        channel.disconnect();
}
```

Další funkcí, kterou třída *SSHDriver* má, je spouštění vzdálených příkazů na clusteru. K tomu se používá metoda *runExec()*. Opět dojde k navázání spojení s Hydrou a vytvoření komunikačního kanálu. Zde však není použit kanál *ChannelSftp*, nýbrž *ChannelExec*. Tento kanál umožňuje získat chybové a datové proudy, které jsou výsledkem příkazů spuštěných na clusteru. Chybový proud je čten pomocí třídy *InputStream*. Ten může být následně analyzován a může sloužit k zobrazení chybových hlášení z clusteru přímo na lokálním počítači.

Kód pro inicializaci a nastavení takové komunikace je zobrazen ve zjednodušené formě na následujících řádcích:

```
Channel channel = session.openChannel("exec");
((ChannelExec) channel).setCommand(command);
channel.setInputStream(null);
((ChannelExec) channel).setErrStream(System.err);
InputStream in = channel.getInputStream();
channel.connect();
byte[] tmp = new byte[1024];
while (true) {
    while (in.available() > 0) {
    int i = in.read(tmp, 0, 1024);
    if (i < 0)
        break;
}
```

V této části kódu je nejprve nastaven komunikační kanál ke vzdálenému ovládání příkazové řádky. Také se nastavují datové a chybové proudy. Po jejich nastavení může být spojení navázáno. *InputStream* je testován metodou *available()*. V případě, kdy proud neobsahuje žádná data, vrací tato metoda 0. V opačném případě je celý proud přečten a s jeho obsahem je možné dále pracovat.

2.3.3 Uložení uživatelského nastavení

Aby uživatel nemusel pokaždé zadávat heslo pro připojení k výpočetnímu clusteru, je v aplikaci možnost si heslo uložit. V opačném případě je uživatel na heslo dotázán při spouštění každého z výpočtů. Nastavení týkající se připojení ke clusteru je uloženo do textového souboru. Z těchto důvodů je potřeba zajistit, aby si heslo nepřečetla neopráv-

něná osoba. Jeden z prvních návrhů bylo uložení celého nastavení do komprimovaného souboru, který by byl chráněn heslem. Po nastudování přístupu jiných programů (např. Total Commander) k této problematice bylo za vyhovující shledáno řešení jiné. Heslo je zašifrováno a v zašifrované podobě uloženo do stejného souboru jako zbytek nastavení.

Pro šifrování a dešifrování heslových frází slouží třída *Encryptor*. Tato třída využívá balíčky *java.security* a *javax.crypto*. Jako šifrovací algoritmus je použit AES. Jedná se o symetrickou blokovou šifru, která šifruje i dešifruje stejným klíčem data rozdělená do bloku pevně dané délky. Velikost klíče v tomto případě je 128 bitů.

Jeden z problémů tohoto řešení je generování klíče pro šifrování. V případě, kdy by byl klíč pevně definovaný, šlo by kopírovat soubory s nastavením mezi počítači. Dále by bylo možné heslovou frázi odvodit. Proto je klíč generován v závislosti na počítači a uživateli, který je aktuálně přihlášen. Fráze je pak určitou kombinací těchto dvou hodnot. Kód pro inicializaci *Encryptoru*, šifrování, dešifrování a generování heslové fráze vypadá následovně:

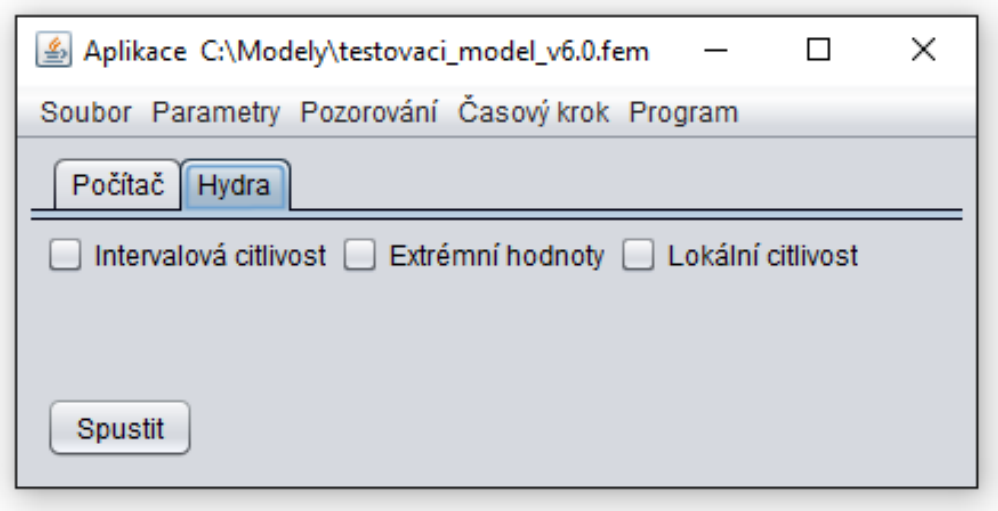
```
private void init() throws Exception {
    klicStr = generujKlic();
    if (aesKlic == null) {
            aesKlic = new SecretKeySpec(klicStr.getBytes(), "AES");
            sifra = Cipher.getInstance("AES");
        }
}
public String sifruj(String text) throws Exception {
        init();
        sifra.init(Cipher.ENCRYPT_MODE, aesKlic);
        return hexString(sifra.doFinal(text.getBytes()));
}
public String desifruj(String text) throws Exception {
        init();
        sifra.init(Cipher.DECRYPT_MODE, aesKlic);
        return new String(sifra.doFinal(byteArray(text)));
}
```

```
private String generujKlic() throws UnknownHostException{
        java.net.InetAddress pc = java.net.InetAddress.getLocalHost();
        return (System.getProperty("user.name") +
        pc.getHostName()).substring(0, 16);
}
```

Metoda *init()* slouží k inicializaci *Encryptoru*. Nejprve je vygenerována heslová fráze pro šifrování. Dále je nastaven šifrovací algoritmus. Jak už bylo napsáno výše, jedná se o algoritmus AES. Metoda *sifruj* provede inicializaci, následně šifrování a vrátí zašifrovaný řetězec. Obdobně funguje i metoda *desifruj*, která vrací řetězec v dešifrované podobě. Jako poslední je uvedená metoda *generujKlic*. Ta pomocí třídy *InetAddress* zjistí název počítače. Ten je pak připojen ke jménu přihlášeného uživatele. Takto vytvořený klíč je pak oříznut na 16 znaků a vrácen jako nová heslová fráze.

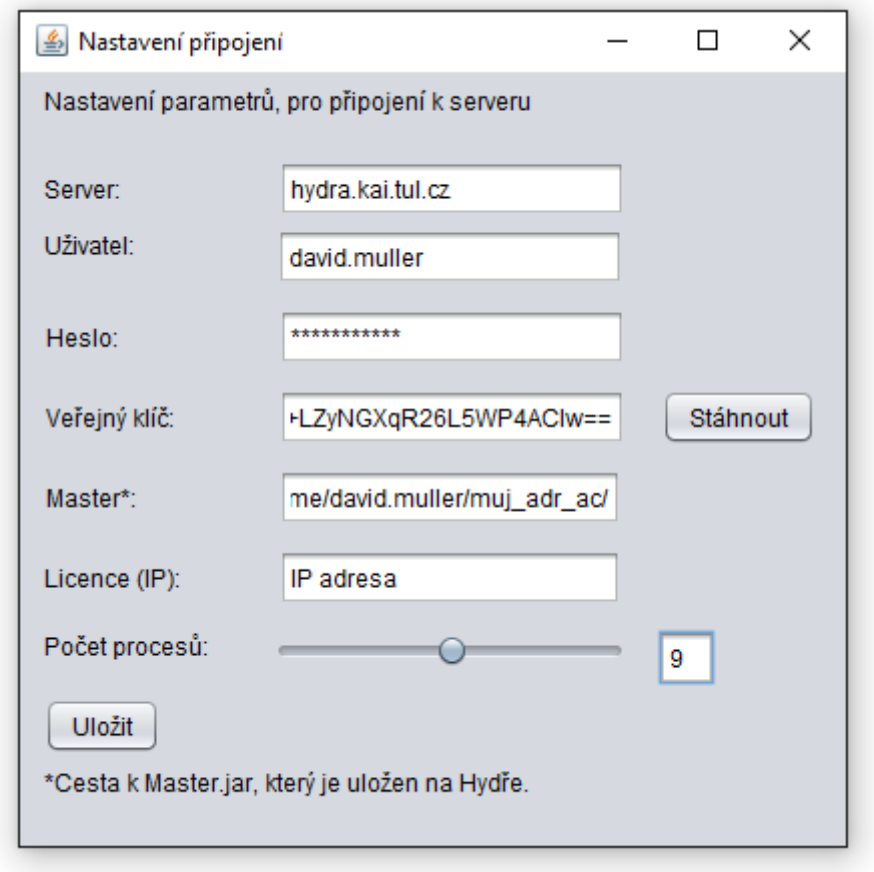
2.4 Rozšíření grafické části

Kvůli novým funkcionalitám byla původní grafická část aplikace Analyza_citlivosti rozšířena. Toto rozšíření se týká hlavního okna a jedné položky v menu Soubor. V hlavním okně aplikace přibyla záložka Hydra, kde lze spustit výpočty právě za pomocí výpočetního clusteru Hydra, který je zmíněn v kapitole *1.2.1 Výpočetní cluster Hydra*. Na *Obrázku 3 Rozšíření hlavního okna aplikace* je vidět, že na Hydře lze analyzovat všechny druhy citlivostí, které byly popsány v kapitole *1.4 Druhy citlivostí*. Tyto možnosti se však objeví až po nahrání některého z modelů do programu.



Obrázek 3 Rozšíření hlavního okna aplikace

Druhá změna se tedy týká položky v menu Soubor. Přibyla zde položka s názvem *Nastavení pro Hydru*. Po jejím rozkliknutí je uživateli zobrazeno okno, které umožňuje kompletní nastavení připojení k Hydře. Pro správnou funkci je zde potřeba vyplnit všechny požadované náležitosti. Jde především o název serveru, uživatele, heslo, cestu k Master.jar souboru a pracovní složku, adresu licenčního serveru a počet výpočetních procesorů. Poslední ještě nezmíněnou položkou je veřejný klíč serveru. Ten je stažen automaticky po stisku tlačítka *Stáhnout*. Toto tlačítko funguje jenom v případě, že je vyplněn správný název serveru, uživatelské jméno a heslo. Klíč je stažen, ale před jeho uložením je uživatel dotázán, zda tento nově stažený klíč chce uložit či nikoliv. V případě, že se klíč z nějakého důvodu nepodaří stáhnout, je o této skutečnosti uživatel informován dialogovým oknem s chybovou hláškou. Okno s nastavením lze vidět na *Obrázku 4 Nastavení pro Hydru*.



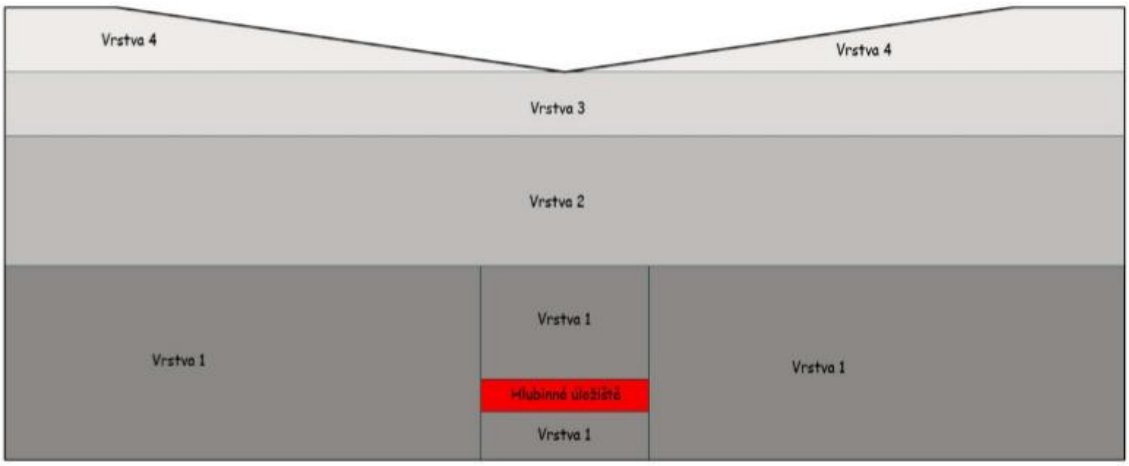
Obrázek 4 Nastavení pro Hydru

3 Testování programu

Tato kapitola se zabývá testováním výsledné aplikace. Je zde popsán hydrogeologický model, který slouží k testování. Tato část práce obsahuje volbu parametrů a pozorování daného modelu. Dále je zde uvedeno srovnání s výsledky práce pana Krause [13], srovnání dob výpočtu při použití různých počtů procesorů. Jako poslední se kapitola zabývá interpretací výsledků lokální a intervalové citlivosti.

3.1 Popis modelu

Pro testování byl využit model vytvořený v práci pana Krause [13] a to zejména proto, aby bylo možné některé z vypočtených citlivostí reprodukovat a výsledky ověřit. Model popisuje proudění podzemní vody, která je do země vsakována v důsledku srážek, prochází podzemním úložištěm (např. jaderného odpadu) a opět se vrací na povrch. Model má na šířku 2000m a 700m na výšku. Modelem protéká řeka a to ve výšce 620m. Model s popisem jednotlivých vrstev je zobrazen na *Obrázku 5 Rozložení testovacího modelu do vrstev s různou propustností*.



Obrázek 5 Rozložení testovacího modelu do vrstev s různou propustností (Zdroj: [13])

Model je diskretizován do trojúhelníků. Těchto trojúhelníků obsahuje přesně 1000. Dále bylo nastaveno saturované prostředí, ustálené proudění a neustálený transport. Protože uvažujeme pouze dvourozměrný model, je jeho zobrazení nastaveno jako vertikální. Jako poslední obecné nastavení je volba časového kroku. Ten je nastaven jako automatický a jako koncový čas je nastavena hodnota $365x10^6$ dní.

Dále jsou modelu nastaveny počáteční podmínky. Jedná se o propustnost jednotlivých vrstev a pórovitost celé oblasti. Jako další jsou nastaveny okrajové podmínky. Konkrétně to jsou úhrn srážek[5], piezometrická výška a koncentrace. Projekt obsahoval více variant nastavení propustnosti, ale pro účely této práce byla vybrána verze číslo 1. Nastavení všech parametrů je uvedeno v *Tabulka 2 Nastavení parametrů modelu.*

Tabulka 2 Nastavení parametrů modelu

Vrstva	Propustnost [10^{-4}m/s]	Pórovitost	Koncentrace [mg/l]
Úložiště	10^{-7}	0,3	0,1/1/10
Vrstva 1	10^{-9}	0,3	0
Vrstva 2	10^{-9}	0,3	0
Vrstva 3	10^{-7}	0,3	0
Vrstva 4	10^{-7}	0,3	0

3.2 Porovnání výsledků

Jak už bylo zmíněno, tato práce využívá jen jednu variantu modelu, který byl převzat z projektu pana Krause [13]. Proto jsou porovnávány pouze výsledky s touto variantou. Nejprve byly porovnávány zjištěné hodnoty pozorování v uzlu číslo 67 a v koncovém časovém kroku. Naměřené hodnoty maximální koncentrace uvedené v *Tabulce 3 Výsledné*

[5] Úhrn srážek je definován nad oblastí znázorněné na *Obr. 1 Definice oblasti pro okrajovou podmínku úhrnu srážek* (Zdroj: aplikace FEFLOW) a nabývá hodnot 100/400/500 [mm/rok]

hodnoty maximální koncentrace a hodnoty časů, kdy tato maxima nastala včetně volby jednotlivých parametrů se rovnají naměřeným hodnotám v projektu. Naměřený čas, ve kterém toto maximum nastalo, se však s projektem nepatrně liší, což může být způsobeno automatickou volbou časového kroku. Výsledek výpočtu aplikace byl proto ručně ověřen a zjištěná doba, při které maximum nastalo, odpovídá výsledku výpočtu aplikace.

Tabulka 3 Výsledné hodnoty maximální koncentrace a hodnoty časů, kdy tato maxima nastala včetně volby jednotlivých parametrů

Koncentrace	Úhrn srážek [mm/rok]	Hodnota maximální koncentrace [mg/l]	Čas maximální koncentrace [den]
0,1	100	0,000724	19 626 300
0,1	400	0,000773	4 675 712
0,1	500	0,000777	3 759 424
1	100	0,007246	19 652 150
1	400	0,007738	4 694 040
1	500	0,007782	3 777 456
10	100	0,0724	19 725 960
10	400	0,0776	4 751 690
10	500	0,0780	3 831 775

Dále byly porovnávány výsledky vypočtené citlivosti. Jako vhodná citlivost pro porovnání mezi projektem Krause a touto prací byla zvolena citlivost intervalová. Její škálování totiž odpovídá škálování v projektu. Bohužel do tabulky vypočtené citlivosti maximální koncentrace na úhrn srážek byly v projektu dosazeny špatné hodnoty maximální koncentrace v uzlu číslo 67 a koncovém časovém kroku, a tak její výsledky neodpovídají výsledkům vypočteným aplikací. Z těchto důvodů byla tabulka přepočítaná a až poté mohly být výsledky porovnávány. Přepočítané tabulky z projektu jsou uvedeny v *Příloze B: Dodatek k porovnání s projektem Krause*. Výsledné hodnoty včetně volby parametrů, při kterých byly tyto hodnoty zjištěny, jsou zaznamenány v *Tabulce 4 Porovnání výpočtů citlivosti maximální koncentrace v uzlu 67 na úhrn srážek*.

Rozdíl ve znaménku zjištěných hodnot mezi projektem a aplikací je způsoben faktem, že aplikace při škálování používá navíc absolutní hodnotu. Dále si lze všimnout nepatrné odchylky mezi výsledky citlivostí. Tato odchylka je způsobena zaokrouhlováním, kdy aplikace využívá pro výpočet více desetinných míst než autor projektu.

Tabulka 4 Porovnání výpočtů citlivosti maximální koncentrace v uzlu 67 na úhrn srážek

Úhrn srážek [mm/rok]	-100	-400	-100	-500	-400	-500
Propustnost [10^{-4}m/s]	Tabulka 3	Tabulka 3	Tabulka 3	Tabulka 3	Tabulka 3	Tabulka 3
Koncentrace v úložišti [mg/l]	1	1	1	1	1	1
Maximální koncentrace v uzlu 67	0,007246	0,007738	0,007246	0,007782	0,007738	0,007782
Citlivost (Projekt)	0,054725		0,053500		0,025515	
Citlivost (Aplikace)	-0,054803		-0,053570		-0,025509	

K dalšímu porovnání byly použity výsledky citlivosti rychlosti vyplavení maximální koncentrace na úhrn srážek. V této tabulce projektu pana Krause se bohužel vyskytla chyba při výpočtu střední hodnoty výsledných časů. Proto musela být před porovnáním tabulka opět upravena. Dále bylo zjištěno, že výsledné časy dosažení maximální koncentrace odpovídají časům, kdy byl parametr koncentrace v úložišti nastaven na hodnotu 0,1 mg/l nikoliv 1 mg/l, jak uvádí tabulka v projektu. Přepočítané tabulky z projektu jsou uvedeny v *Příloze B: Dodatek k porovnání s projektem Krause*. *Tabulka 5 Porovnání výpočtů citlivosti doby vyplavení maximální koncentrace v uzlu 67 na úhrn srážek* obsahuje tedy přepočítanou citlivost projektu a také citlivost vypočítanou samotnou aplikací. Lze si všimnout, že výsledné hodnoty se opět liší znaménkem a drobnou odchylkou. Tento fakt opět souvisí se zaokrouhlováním, použitou absolutní hodnotou a automatickou volbou časového kroku.

Tabulka 5 Porovnání výpočtů citlivosti doby vyplavení maximální koncentrace v uzlu 67 na úhrn srážek

Úhrn srážek [mm/rok]	-100	-400	-100	-500	-400	-500
Propustnost [10^{-4}m/s]	Tabulka 3	Tabulka 3	Tabulka 3	Tabulka 3	Tabulka 3	Tabulka 3
Koncentrace v úložišti [mg/l]	0,1	0,1	0,1	0,1	0,1	0,1
Čas, kdy nastalo maximum koncentrace [rok]	53 698	12 821	53 698	10 301	12 821	10301
Citlivost (Projekt)	-1,024193		-1,017133		-0,980884	
Citlivost (Aplikace)	1,025333		1,017728		0,977648	

3.3 Srovnání rychlostí výpočtu

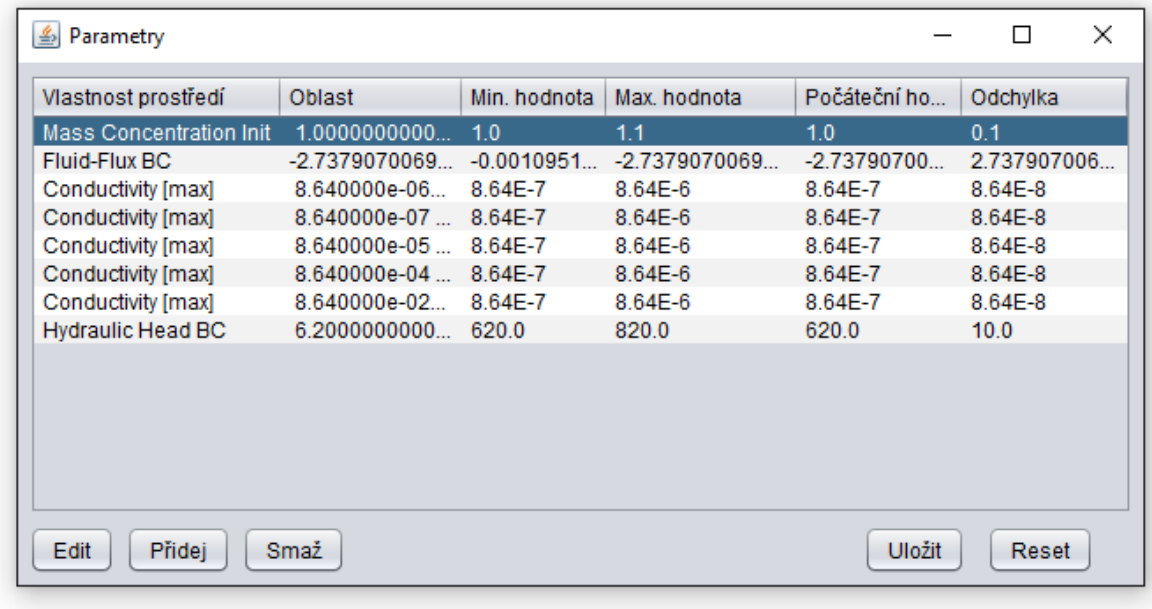
Tato kapitola se zabývá volbou parametrů a pozorování pro testovací model. Je zde uvedeno srovnání doby výpočtu při spuštění aplikace na různém počtu procesorů. Pro zajímavost jsou v kapitole uvedeny i doby výpočtů jednotlivých citlivostí na lokálním počítači. Všechny časy jsou pouze orientační, protože doba výpočtu závisí také na aktuálním zatížení clusteru a vytížení počítačové sítě. Aby bylo zachováno pojmosloví mezi aplikací pro analýzu citlivosti, aplikací FEFLOW a touto prací, nejsou v této kapitole názvy parametrů překládány.

3.3.1 Volba parametrů

Pro srovnání rychlostí při výpočtu na různých počtech procesorů bylo potřeba přidat další parametry. V případě, kdybychom měli pouze jeden parametr, proběhly by na Hydře pouze dva výpočty. Je zřejmé, že v takovém případě by rychlost výpočtu na dvou procesorech byla stejná jako na deseti, protože osmi procesorům by nebyl přidělen žádný z výpočtů.

Dále bylo zjištěno, že pokud byl pro citlivost použit pouze jeden parametr, byl výpočet na lokálním počítači zhruba 14krát rychlejší než za použití Hydry. Tento fakt je způsoben připojováním k Hydře, kopírováním modelu na Hydru, velkou režií při spouštění a ukončování MPI démonů a stahováním výsledků. I proto se jeví jednotné grafické rozhraní pro spuštění výpočtů na lokálním počítači a na clusteru jako vhodná volba. V případě jednoduchých modelů s málo parametry má smysl použít lokální počítač. Pokud jsou však modely složité, jejich výpočet trvá řádově hodiny. Přidá-li se k tomu ještě velké množství parametrů, trvalo by zpracování na lokálním počítači podstatně delší dobu. Samozřejmě za předpokladu odpovídajícího množství workerů (více jak dva).

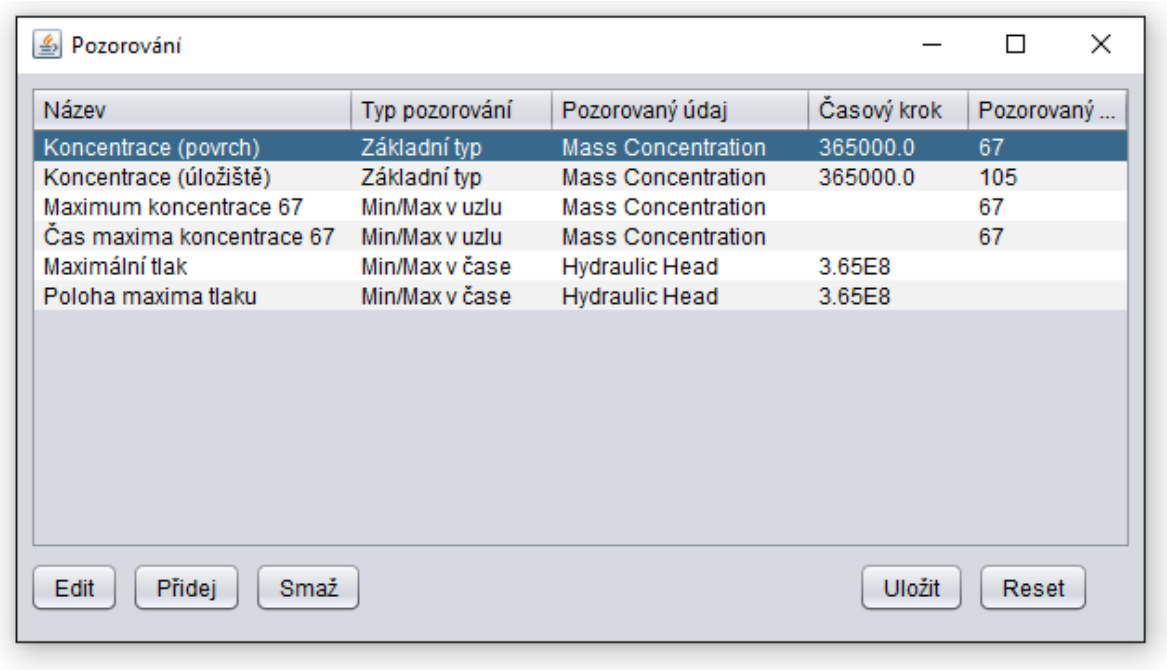
Vhodnými parametry se jeví propustnosti jednotlivých vrstev modelu, dále změny koncentrace v úložišti, piezometrická výška v oblasti, kde protéká řeka a již testovaný úhrn srážek. Definice oblastí pro jednotlivé parametry obsahuje *Příloha A: Nastavení parametrů*. Konkrétní nastavení jednotlivých parametrů lze vidět na *Obrázku 6 Nastavení parametrů*. Nastavení parametru Conductivity (propustnosti) je zde ve čtyřech variantách. Tyto varianty odpovídají čtyřem vrstvám, které jsou vidět na *Obrázku 5 Rozložení testovacího modelu do vrstev s různou propustností*. První parametr propustnosti odpovídá vrstvě číslo čtyři, druhý vrstvě číslo tři a tak dále.



Obrázek 6 Nastavení parametrů

3.3.2 Volba pozorování

Protože úloha pana Krause zkoumala vliv parametrů na vyplavování koncentrace nějaké blíže nespecifikované látky z úložiště, byla pozorování v této práci volena obdobným způsobem. Jedná se především o analýzu citlivosti koncentrací na povrchu a přímo v úložišti, dále citlivost maximální koncentrace v uzlu číslo 67 a citlivost piezometrické výšky v úložišti. Nastavení jednotlivých pozorování je uvedeno na *Obrázku 7 Nastavení pozorování*.



Obrázek 7 Nastavení pozorování

3.3.3 Srovnání rychlosti výpočtu při různém počtu výpočetních uzlů

Pro srovnání rychlosti výpočtu na různých počtech procesorů byl použit model definovaný v kapitole *3.1 Popis modelu*. O nastavení parametrů a pozorování pojednávají kapitoly *3.3.1 Volba parametrů* a *3.3.2 Volba pozorování*. Výsledné srovnání časů výpočtu jednotlivých citlivostí při použití různého počtu procesorů je uvedeno v *Tabulce 6 Srovnání doby výpočtu při použití různých počtů procesorů*.

Tabulka 6 Srovnání doby výpočtu při použití různých počtů procesorů[6]

Citlivost	2 procesory	4 procesory	8 procesorů	16 procesorů
Lokální citlivost	68s	38s	29s	-
Intervalová citlivost	120s	57s	35s	-
Extrémní hodnoty	454s	228s	119s	65s

Hodnoty doby výpočtu lokální a intervalové citlivosti při použití 16ti procesorů nejsou zaznamenány proto, že použití 16ti procesorů při sedmi parametrech a tedy osmi výpočtech by nevedlo k žádnému zrychlení. Z tabulky je patrné, že dvojnásobný počet procesorů odpovídá zhruba poloviční době, která je pro výpočet potřeba. Jelikož jsou uzly na Hydře vybaveny různými procesory (viz kapitola *1.2.1 Výpočetní cluster Hydra*) je doba výpočtu ovlivněna i uzly, na kterých se výpočet spustí. Dalšími faktory ovlivňujícími délku výpočtu jsou aktuální zatížení sítě, nastavení parametrů a pozorování a samozřejmě i hydrogeologický model, který je spouštěn. Jako zajímavé se jeví i srovnání doby výpočtu při spuštění na lokálním počítači. Zde však záleží na hardwarovém vybavení tohoto počítače. Výsledné časy jsou zaznamenány v *Tabulce 7 Srovnání doby výpočtu různých citlivostí na lokálním počítači*.

Tabulka 7 Srovnání doby výpočtu různých citlivostí na lokálním počítači

Lokální citlivost	Intervalová citlivost	Extrémní hodnoty
27s	49s	481s

3.4 Analýza výsledků jednotlivých citlivostí

Tato kapitola se zabývá výsledky analýzy citlivosti testovacího modelu zmíněného v kapitole *3.1 Popis modelu*. Výsledky jsou vyhodnocovány pro citlivost lokální a intervalovou. Dále je zde uvedená možná interpretace těchto zjištěných hodnot nebo přínos, který by pro pozorovatele mohly mít.

[6] Počet procesorů v tabulce odpovídá počtu workerů. Skutečný počet procesorů využitých k výpočtu je tedy vždy o jedna vyšší.

3.4.1 Lokální citlivost

V případě této citlivosti ukazují výsledky citlivosti pozorování jen na minimální změnu hodnoty parametru. Velký vliv na daná pozorování měl parametr Mass Concentration Init. Jedná se o počáteční podmínku koncentrace. Tento parametr měl velký vliv na pozorování s názvem Koncentrace (povrch), Koncentrace (úložiště) a Maximum koncentrace 67. Tento fakt není nikterak překvapující, protože změna počáteční koncentrace by měla ovlivňovat i hodnoty koncentrace na konci výpočtu. Dalšími velmi vlivnými parametry byly propustnosti jednotlivých vrstev modelu. Ty měly opět největší vliv na hodnoty koncentrací a to zejména na povrchu. Vybrané výsledky lokální citlivosti jsou zobrazeny v *Tabulce 8 Výsledky lokální citlivosti*.

Tabulka 8 Výsledky lokální citlivosti

Parametr	Pozorování	Citlivost
Mass Concentration Init	Koncentrace (povrch)	1,000258
Mass Concentration Init	Koncentrace (úložiště)	1,000478
Mass Concentration Init	Maximum koncentrace 67	1,055837
Conductivity [max] - vrstva 4	Koncentrace (povrch)	-0,368409
Conductivity [max] – vrstva 3	Koncentrace (povrch)	-0,560878
Conductivity [max] – vrstva 2	Koncentrace (povrch)	-0,293699
Conductivity [max] – vrstva 1	Koncentrace (povrch)	1,064739
Conductivity [max] – úložiště	Koncentrace (povrch)	0,092433

3.4.2 Intervalová citlivost

Z výsledků intervalové citlivosti lze vyčíst, že nejvýznamnějším parametrem ovlivňujícím hodnotu koncentrace v úložišti je propustnost. V rámci této citlivosti se však výsledné hodnoty pro svrchní tři vrstvy až na drobné odchylky rovnají. Jako další parametr, který výrazně ovlivňuje toto pozorování, lze shledat počáteční podmínku koncentrace. Tyto dva parametry výrazně ovlivňují i hodnoty koncentrací na povrchu. Vybrané hodnoty intervalové citlivosti jsou zobrazeny v *Tabulce 9 Výsledky intervalové citlivosti*.

Tabulka 9 Výsledky intervalové citlivosti

Parametr	Pozorování	Citlivost
Mass Concentration Init	Koncentrace (povrch)	1,000245
Mass Concentration Init	Koncentrace (úložiště)	1,000455
Mass Concentration Init	Maximum koncentrace 67	1,053037
Fluid-Flux BC	Koncentrace (úložiště)	1,351399
Conductivity [max] - vrstva 4	Koncentrace (úložiště)	0,243230
Conductivity [max] – vrstva 3	Koncentrace (úložiště)	0,276589
Conductivity [max] – vrstva 2	Koncentrace (úložiště)	0,292339
Conductivity [max] – vrstva 1	Koncentrace (úložiště)	-0,181999
Conductivity [max] – úložiště	Koncentrace (úložiště)	-0,060014

3.5 Srovnání výsledků lokálních výpočtů a výpočtů na clusteru

Při analýze tabulek jednotlivých citlivostí, které byly vypočítány na lokálním počítači, bylo zjištěno, že se tyto tabulky nepatrně liší od tabulek, které obsahovaly čísla výpočtů poskytnutých clustrem. Tento fakt lze přisoudit tomu, že jednotlivé počítače mohou pracovat s různou přesností. Nestandardním chováním však byl nepatrně odlišný výstup v případě, kdy byl dvakrát po sobě spuštěn stejný model a to na stejném výpočetním uzlu clusteru. Takové chování má samozřejmě vliv na výsledné hodnoty. Dále se v některých případech, po zavolání FEFLOW z příkazové řádky, FEFLOW vůbec nespustilo. Tento problém byl vyřešen tím, že pro chybějící výpočet aplikace vrátí hodnoty NaN.

Závěr

Tato práce se zabývala paralelizací nástroje pro počítačovou podporu analýzy citlivosti hydrogeologických modelů. Jak již bylo zmíněno v úvodu, kvůli rozsahu práce nebyla vysvětlena problematika hydrogeologických modelů či odvozování rovnic uskutečňujících simulaci proudění podzemní vody, transportu hornin a transportu tepla porézním prostředím. Cílem práce bylo paralelizovat program pro analýzu citlivosti, který umí analyzovat celkem tři druhy těchto citlivostí. Jedná se především o citlivost lokální, kterou se zabývá kapitola *1.4.1 Lokální citlivost*. Dále je to intervalová citlivost, která je definovaná v kapitole *1.4.2 Intervalová citlivost*. Jako poslední jsou extrémní hodnoty, o kterých pojednává kapitola *1.4.3 Extrémní hodnoty*. Všechny tři typy citlivostí umožňují uživateli získat informace o dějích, které uvnitř modelu probíhají. Je však dobré si uvědomit, že analýza citlivosti nachází uplatnění nejen v oblasti hydrogeologických modelů, ale má uplatnění v řadě dalších oborů, které s modelováním souvisí.

Vytvořená aplikace, která je hlavním přínosem práce, je schopna počítat citlivost lokální a intervalovou a také extrémní hodnoty paralelně a výsledky poskytnout ve formě tabulek uživateli. To, že jsou výpočty spouštěny paralelně na výpočetním clusteru, značně zkracuje dobu potřebnou pro analýzu a umožňuje provádět analýzu citlivosti složitých hydrogeologických modelů s mnoha parametry v rozumném čase. Je však nutné poznamenat, že připojování se ke clusteru a spouštění běhového prostředí pro paralelní aplikace nějakou dobu trvá a pro některé modely, které mají navíc málo parametrů, je výpočet za použití lokálního počítače rychlejší. Avšak s rostoucím počtem parametrů a u složitějších modelů je přínos paralelního výpočtu velký.

V budoucnu by bylo vhodné otestovat aplikaci za použití jiného clusteru nejen Hydry. Dále rozšířit aplikaci tak, aby umožňovala práci i s jinými nástroji pro simulaci proudění podzemní vody, transportu rozpuštěných látek a transportu tepla. To zejména proto, že je aplikace v současné době závislá na aplikaci FEFLOW. Bez ní nelze provádět výpočty a analýza citlivosti bez ní není možná. Dalším možným rozšířením by bylo umožnit výpočet dalších citlivostí, které by mohly mít u některých modelů lepší vypovídající hodnotu. Cíl práce, kterým bylo paralelizovat nástroj pro počítačovou podporu analýzy citlivosti hydrogeologických modelů, považuji za splněný.

Terminologický slovník

Termín	Význam (zdroj)
AES	AES je bloková symetrická šifra. (autor)
C++	C++ je hojně využívaný, kompilovaný programovací jazyk. (autor)
Diskretizace	Jedná se o nahrazení spojitého prostředí systémem diskrétních bodů, v nichž jsou definovány parametry popisující vlastnosti konkrétního místa v dříve spojitém prostředí. (autor)
Garbage collector	Jedná se o speciální algoritmus pro automatickou správu paměti. (autor)
GUI	GUI je grafické rozhraní aplikace. (autor)
JSCH	JSCH je knihovna implementující SSH pro programovací jazyk Java.
JavaScript	Jedná se o skriptovací jazyk.
MPI	MPI je standard pro paralelní aplikace a knihovny. (autor)
SSH	SSH je zabezpečený komunikační protokol. (autor)
SPMD	SPMD (Single program, multiple data) je jeden ze základních modelů dekompozice paralelních aplikací. (autor)
Piezometrická výška	Označuje výšku, kam vystoupá hladina podzemní vody, je-li proveden vrt do napjaté zvodně. (autor)
PHP	Je skriptovací programovací jazyk, který je určený především k vytváření dynamických internetových stránek. (autor)

Seznam literatury

[1] ORACLE CORPORATION. Vítejte u NetBeans. In: Welcome to NetBeans [online]. © 2013 [cit. 2016-01-19]. Dostupné z: https://netbeans.org/index_cs.html

[2] MACROMEDIA. Java history. In: Papa.det.uvigo.es [online]. 2005 [cit. 2016-02-08]. Dostupné z: http://papa.det.uvigo.es/~theiere/cursos/Curso_Java/history.html

[3] Java Garbage Collection Basics. Oracle|Integrated Cloud Applications & Platform Services [online]. Oracle, 2016 [cit. 2016-01-25]. Dostupné z: http://www.oracle.com/webfolder/technetwork/tutorials/obe/java/gc01/index.html

[4] Detailní technické informace o clusteru Hydra. Hydra/techdetails – Ústav nových technologií a aplikované informatiky [online]. 2010 [cit. 2016-02-15]. Dostupné z: http://www.nti.tul.cz/cz/Hydra/techdetailshttp://www.sciencedirect.com/science/article/pii/0167819196000245

[5] GROPP, William, Ewing LUSK, Nathan DOSS a Anthony SKJELLUM. A high-performance, portable implementation of the MPI message passing interface standard. Parallel Computing [online]. 1996, 22(6), 791-792 [cit. 2016-02-28]. DOI: 10.1016/0167-8191(96)00024-5. ISSN 01678191. Dostupné z: http://linkinghub.elsevier.com/retrieve/pii/0167819196000245

[6] YLONEN a LONVICK. The Secure Shell (SSH) Protocol Architecture [online]. 2006 [cit. 2016-03-09]. Dostupné z: https://tools.ietf.org/pdf/rfc4251.pdf

[7] MÜLLER, David. 2014. Tvorba nástroje pro počítačovou podporu analýzy citlivosti hydrogeologických modelů. Liberec. Bakalářská práce. Technická univerzita v Liberci, Fakulta mechatroniky, informatiky a mezioborových studií. Vedoucí práce Doc.Ing. Jan Šembera, Ph.D.

[8] HILL, Mary C a Claire R TIEDEMAN. Effective groundwater model calibration: with analysis of data, sensitivities, predictions, and uncertainty. Hoboken, N.J.: Wiley-Interscience, 2007, s. 47-50. ISBN 9780471776369.

[9] IBM(R) Distributed Computing Environment Version 3.2 for AIX(R) and Solaris: Application Development Guide - Core Components [online]. IBM Corporation, 2001 [cit. 2016-03-16]. Dostupné z: http://www-01.ibm.com/software/network/dce/library/publications/appdev/html/APPDEV14.HTM

[10] BRADFORD NICHOLS, Dick Buttlar. Pthreads programming. [Nachdr.]. Cambridge [u.a.]: O'Reilly, 1997, s. 31. ISBN 9781565921153.

[11] KRAVAL, Ilja. Design Patterns v OOP: se zaměřením na JAVU, C# a Delphi. 2002, s. 107-111.http://www.jcraft.com/jsch/

[12] JSch - Java Secure Channel [online]. JCraft, Inc., 2014 [cit. 2016-03-16]. Dostupné z: http://www.jcraft.com/jsch/

[13] KRAUS, Vojtěch. Využití analýzy citlivosti v modelování proudění podzemní vody [online]. [cit. 2016-04-15]. Ročníkový projekt. Technická univerzita v Liberci. Vedoucí práce Doc.Ing.Jan Šembera, Ph.D.

Seznam obrázků a tabulek

Textová část

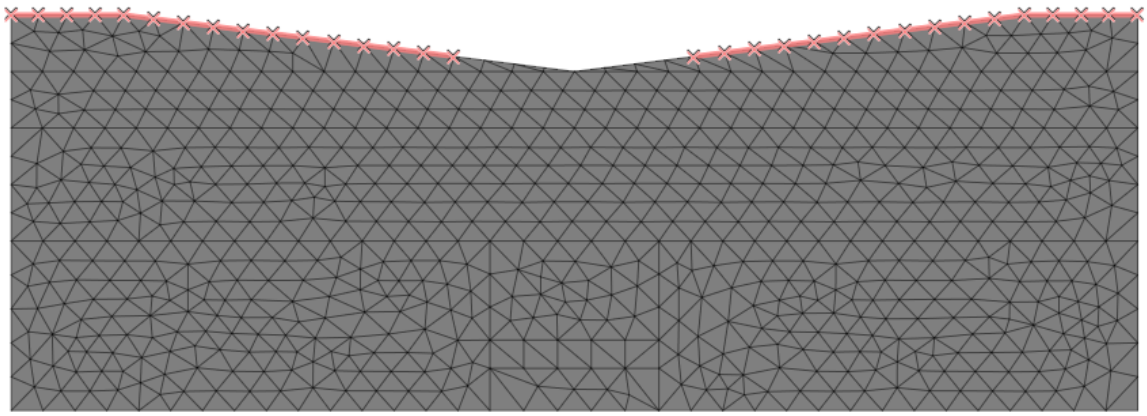
Seznam obrázků

Seznam tabulek

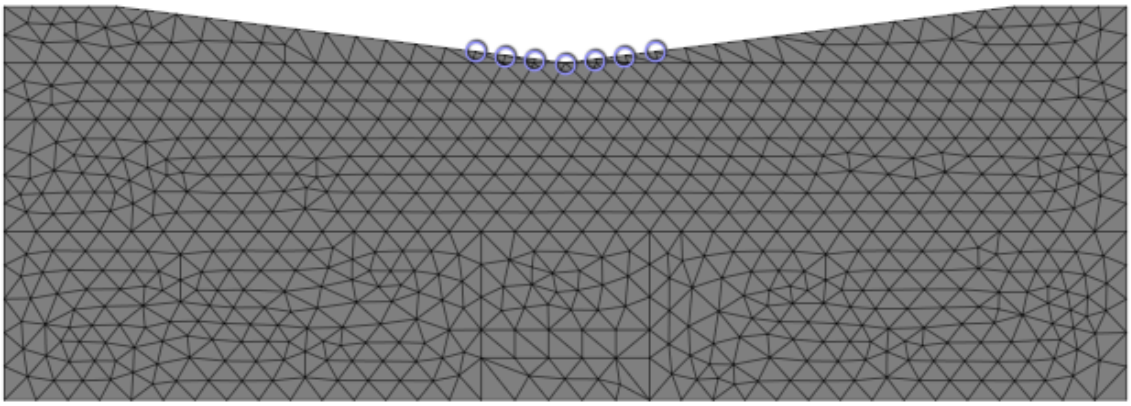
Přílohy

Seznam obrázků

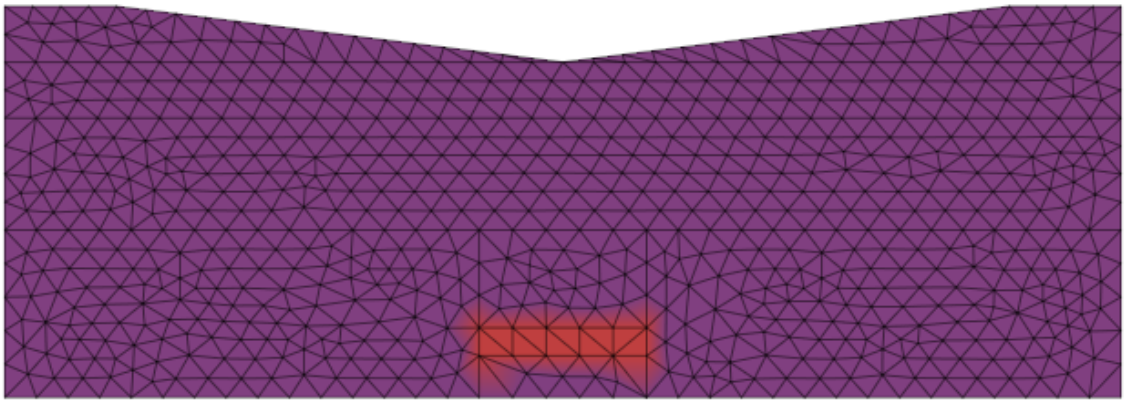
Příloha A: Nastavení parametrů



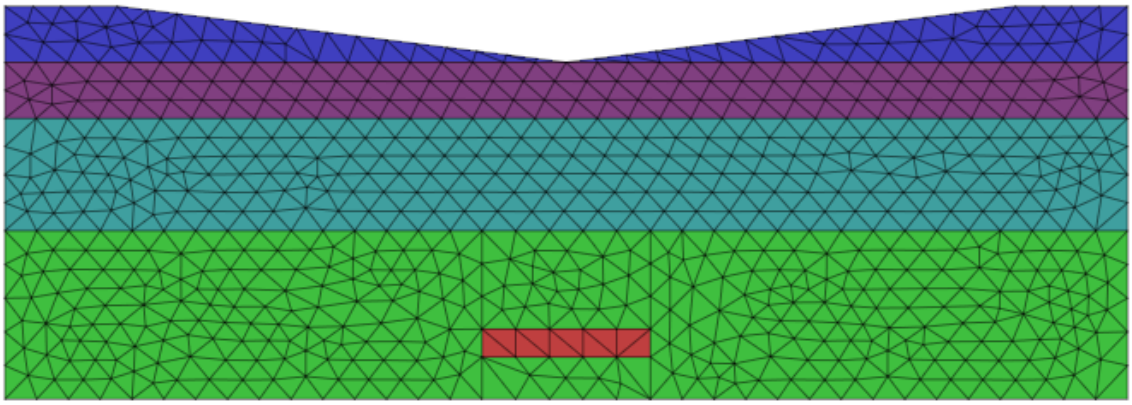
Obr. 1 Definice oblasti pro okrajovou podmínku úhrnu srážek (Zdroj: aplikace FEFLOW)



Obr. 2 Definice oblasti pro okrajovou podmínku piezometrické výšky (Zdroj: aplikace FEFLOW)



Obr. 3 Definice oblasti pro počáteční podmínku koncentrace (Zdroj: aplikace FEFLOW)



Obr. 4 Definice oblastí jednotlivých vrstev propustnosti (Zdroj: aplikace FEFLOW)

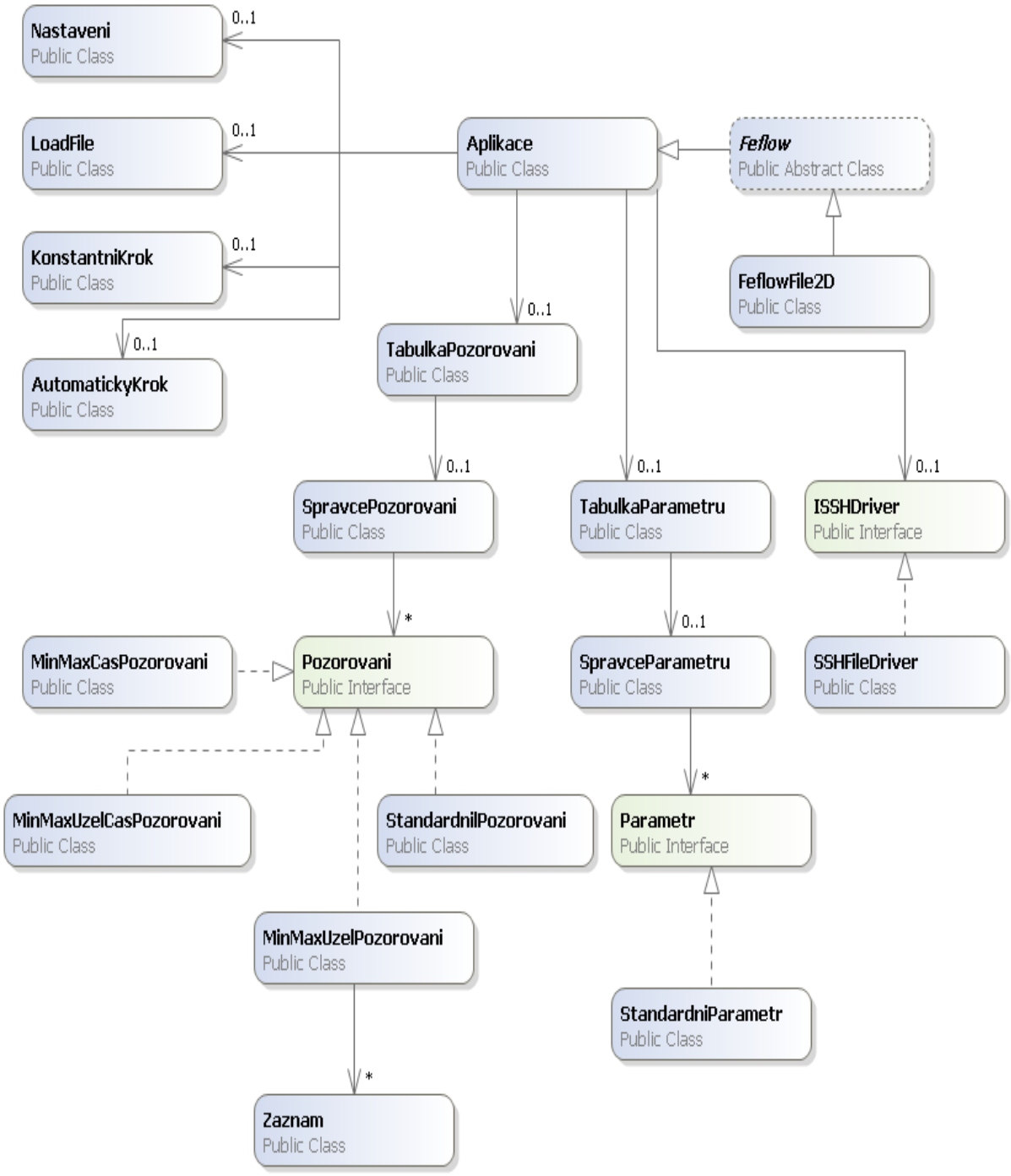
Příloha B: Dodatek k porovnání s projektem Krause

Protože se pan Kraus ve svém projektu (viz [13]) dopustil drobných chyb při kopírování hodnot mezi tabulkami a výpočtu střední hodnoty, bylo za potřebí tyto tabulky pro správné porovnání opravit. Jedná se o tabulky číslo 18 a 21 uvedené pod tímto odstavcem. Aby tabulky přesně odpovídaly tabulkám v projektu, byly upraveny pouze chybné hodnoty. Veškeré hodnoty použité pro výpočet však pocházejí z projektu pana Krause a proto bylo možné tabulky opravit a provést porovnání s výsledky, které vypočítala aplikace. Dále je potřeba zmínit, že pan Kraus počítal s kladnou hodnotou úhrnu srážek, přičemž v modelu byla tato hodnota vyplněna jako záporná.

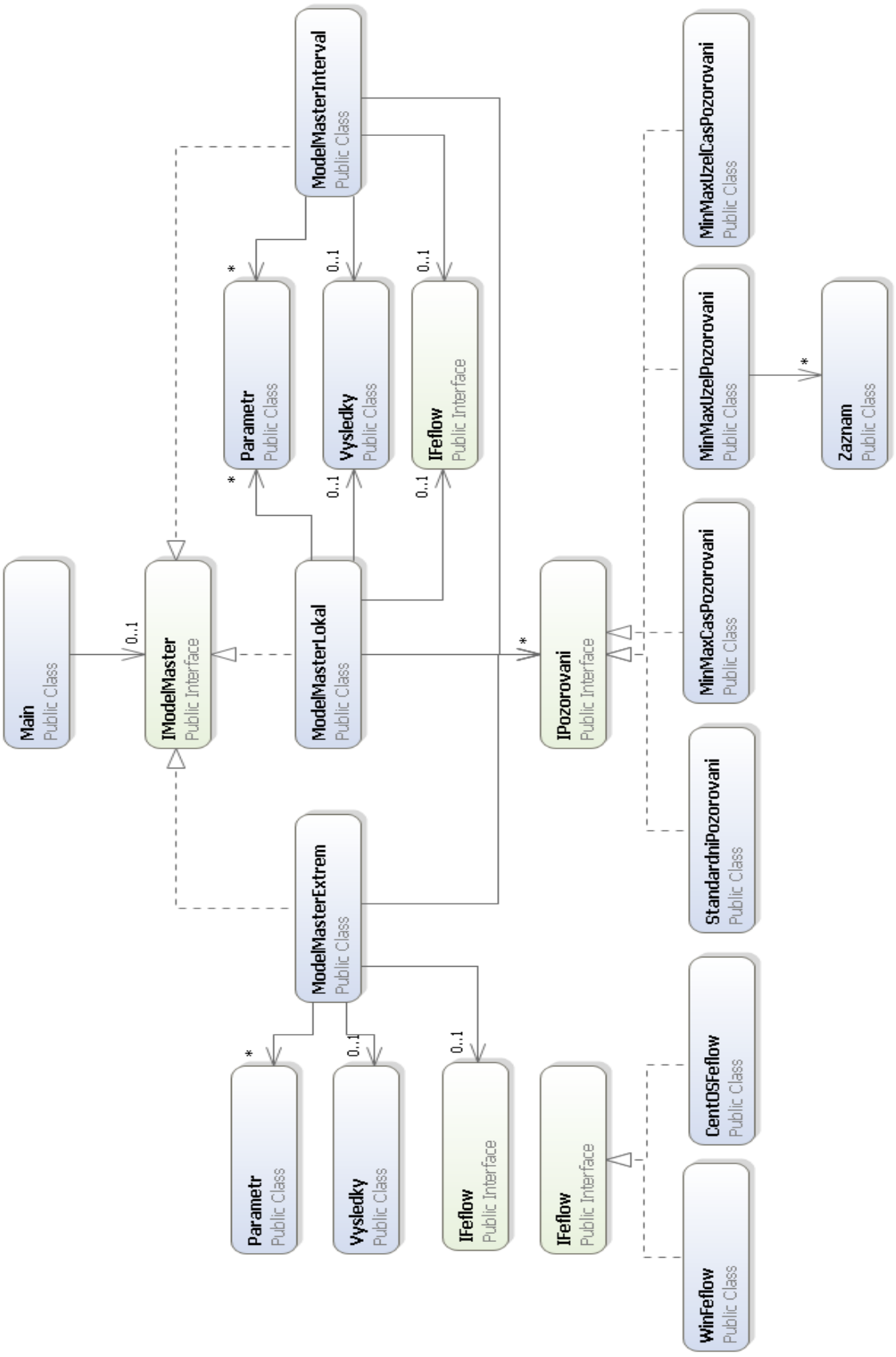
Tabulka 18 - Citlivost úhrnu srážek na maximální koncentraci - varianta 1						
Úhrn srážek [mm/rok]	100	400	100	500	400	500
Propustnost [m/s]	Var. 1	Var. 1	Var. 1	Var. 1	Var. 1	Var. 1
Koncentrace v úložišti [mg/l]	1	1	1	1	1	1
Maximální koncentrace na povrchu [mg/l]	0,007246	0,007738	0,007246	0,007782	0,007738	0,007782
Střední hodnota - koncentrace [mg/l]	0,007492		0,007514		0,00776	
Střední hodnota - úhrnu srážek [mm/rok]	-250		-300		-450	
Citlivost	**0,05472504**		**0,053500133**		**0,025515464**	

Tabulka 21 - Citlivost úhrnu srážek na rychlosti vyplavení - varianta 1						
Úhrn srážek [mm/rok]	100	400	100	500	400	500
Propustnost [m/s]	Var. 1	Var. 1	Var. 1	Var. 1	Var. 1	Var. 1
Koncentrace v úložišti [mg/l]	0,1	0,1	0,1	0,1	0,1	0,1
Doba, za kterou dosáhne max. koncentrace	53698	12821	53698	10301	12821	10301
Střední hodnota - času [roky]	33259,5		31999,5		11561	
Střední hodnota - úhrnu srážek [mm/rok]	-250		-300		-450	
Citlivost	**-1,024193589**		**-1,01713308**		**-0,980884007**	

Příloha C: UML diagram aplikace



Obr. 5 Zjednodušený diagram tříd aplikace Analyza_citlivosti (Zdroj: NClass)



Obr. 6 Zjednodušený diagram tříd aplikace Master (Zdroj: NClass)

Příloha D: Důležité poznámky ke spuštění[7]

D.1 Výpočty na lokálním PC

Pro správnou funkci aplikace, kdy jsou použity pouze lokální výpočty, je potřeba mít v počítači nainstalované FEFLOW a samozřejmě Javu. Dále je potřeba v záložce *Soubor → Nastavení pro počítač* zadat cestu k souboru, kterým se FEFLOW spuští v režimu příkazového řádku (např. *feflow62c.exe*). Poté se v záložce *Soubor → Načíst* vybere model, který má být analyzován. Je potřeba zadat aspoň jeden parametr a jedno pozorování. Následně lze vybrat citlivosti, které mají být spočítány a výpočet spustit.

D.2 Výpočty pomocí clusteru

Aby bylo možné spustit výpočet vzdáleně na clusteru, je zapotřebí mít na tomto clusteru svého uživatele. Dále je potřeba si na tomto clusteru nainstalovat MPJ Express. O jeho instalaci pojednává *Příloha E: MPJ Express*. Po úspěšné instalaci je potřeba nastavit systémovou proměnnou PATH tak, aby obsahovala cestu k souboru, kterým je spouštěn FEFLOW v režimu příkazového řádku. K tomu slouží následující dva příkazy:

```
export FEFLOW_HOME=/cesta/k/feflow/
export PATH=$FEFLOW_HOME/bin:$PATH
```

Tyto řádky je vhodné napsat do souboru *.bashrc* v případě, že výchozí shell je bash. Poté je nutné na cluster nahrát aplikaci Master.jar i s jejími knihovnami.

Následně je potřeba vyplnit veškeré údaje v záložce *Soubor → Nastavení pro Hydru*. V případě, kdy uživatel nevyplní heslo, je na toto heslo dotazován při každém spuštění. Poté už je možné načíst model s parametry a pozorováními a spustit výpočet požadovaných citlivostí.

---

[7] Aplikace je kompatibilní s modely vygenerovanými pro Feflow do verze 6.1.

Příloha E: MPJ Express

E.1 Instalace MPJ Express

Jako první krok je potřeba stáhnout MPJ Express software a rozbalit jej do adresáře, ze kterého bude spouštěn. Dále je nutné nastavit systémovou proměnnou PATH, která obsahuje cesty k adresářům se spustitelnými programy. K tomu slouží následující dva příkazy:

```
export MPJ_HOME=/cesta/k/mpj/
export PATH=$MPJ_HOME/bin:$PATH
```

Aby vše fungovalo správně, je potřeba tyto řádky napsat do souboru *.bashrc* v případě, že výchozí shell je bash. Dále je nutné vytvořit pracovní adresář pro MPJ Express programy. Následně je potřeba zkompilovat MPJ Express knihovnu příkazem:

```
cd $MPJ_HOME; ant
```

E.2 Kompilace testovací aplikace

Otestovat funkčnost MPJ Express lze pomocí jednoduchého programu, který vypíše zprávu z jednotlivých uzlů. Jeho kód je následující:

```
import mpi.*;

public class AhojSvete {
   public static void main(String args[]) throws Exception {
    MPI.Init(args);
    int identifikator = MPI.COMM_WORLD.Rank();
    int velikost = MPI.COMM_WORLD.Size();
    System.out.println("Ahoj z <"+ identifikator+">");
    MPI.Finalize();
    }
}
```

Tento kód musí být následně uložen do souboru AhojSvete.java a zkompilován. Ke kompilaci slouží příkaz:

```
javac -cp .:$MPJ_HOME/lib/mpj.jar AhojSvete.java
```

E.3 Spuštění MPJ Express v cluster-režimu

Pro spuštění aplikace zmíněné v předchozí kapitole je potřeba vytvořit soubor *machines*. V tomto souboru jsou uloženy názvy nebo IP adresy uzlů (počítačů), které může MPJ Express využít. Tento souboru může vypadat například takto:

```
compute-1-6
compute-1-7
compute-1-5
```

Soubor *machines* se musí nacházet v adresáři, ze kterého je spouštěn *AhojSvete.class*, který je výsledkem kompilace programu *AhojSvete.java*. Posloupnost příkazu pro spuštění aplikace je následujicí:

```
cd /home/jmeno.uzivatele/muj_adr/
mpjboot machines
mpjrun.sh -np 3 -dev niodev AhojSvete.class
mpjhalt machines
```

Mpjboot spustí MPJ démona na uzlech, obsažených v souboru *machines*. *Mpjhalt* jej naopak ukončí. *Mpjrun.sh* slouží ke spuštění konkrétní aplikace pomocí MPJ Express. Parametr *–np* určuje počet uzlů, které má aplikace k dispozici. Soubor *machines* může obsahovat seznam několika uzlů, který příkazem *–np* omezime jen na jejich podmnožinu. Parametr –dev určuje zařízení, které má být využito ke komunikaci. V tomto případě je využito niodev (Java New I/O device). Niodev slouží ke spuštění MPJ Express programů na clusterech nebo počítačových sítích.

Aby vše fungovalo, je ještě potřeba přidat právo na spuštění pro soubory mpjboot, mpjhalt, mpjrun.sh. K tomu slouží příkaz:

```
chmod 775 /cesta/mpj/bin/soubor
```

Pokud je vše nastaveno dobře a vše funguje, bude výpis z programu následující:

```
MPJ Express (0.43) is started in the cluster configuration
Ahoj z <2>
Ahoj z <0>
Ahoj z <1>
```

Příloha F: Obsah přiloženého CD

Přiložené CD k diplomové práci je členěno do čtyř adresářů.

- text diplomové práce
  - diplomova_prace_2016_David_Muller.pdf
  - diplomova_prace_2016_David_Muller.doc
- program[8]
  - Analyza_Citlivosti.jar
  - Master.jar
- zdrojový kód programu
  - DP.rar
- model
  - testovaci_model_Muller.fem
  - srovnavaci_model_Kraus.fem

[8] Pro správnou funkci programu je potřeba mít nainstalovaný tzv. interpret programovacího jazyka Java.