**Technická univerzita v Košiciach**
**Fakulta elektrotechniky a informatiky**

# Paralelizácia dvojrozmerných modelov modulácie kozmického žiarenia v heliosfére

**Diplomová práca**

**2020** **Bc. Michal Solanik**

**Technická univerzita v Košiciach**
**Fakulta elektrotechniky a informatiky**

# Paralelizácia dvojrozmerných modelov modulácie kozmického žiarenia v heliosfére

**Diplomová práca**

| | |
|---|---|
| Študijný program: | Informatika |
| Študijný odbor: | 9.2.1. Informatika |
| Školiace pracovisko: | Katedra počítačov a informatiky (KPI) |
| Školiteľ: | RNDr. Pavol Bobík PhD. |
| Konzultant: | doc. Ing. Ján Genči PhD. |

**Košice 2020** **Bc. Michal Solanik**

| | |
|---|---|
| Názov práce: | Paralelizácia dvojrozmerných modelov modulácie kozmického žiarenia v heliosfére |
| Pracovisko: | Katedra počítačov a informatiky, Technická univerzita v Košiciach |
| Autor: | Bc. Michal Solanik |
| Školiteľ: | RNDr. Pavol Bobík PhD. |
| Konzultant: | doc. Ing. Ján Genči PhD. |
| Dátum: | 4. 5. 2020 |
| Kľúčové slová: | CUDA, distribuované systémy, heliosféra, GPGPU |

Abstrakt: Cieľom predchádzajúcej bakalárskej práce bola paralelizácia 1D modelov propagácie častíc kozmického žiarenia v heliosfére. V prípade 1D F-p modelu sa pri malých dĺžkach kroku vyskytla chyba, ktorú sme nazvali pulzácia. Stanovili sme tri hypotézy možnej príčiny výskytu pulzov, ale ani jedna sa nepotvrdila. Po pristúpení ku analýze rozsahu parametrov prítomnosti pulzácie, sme zistili, že pulzácie sa vyskytujú v 1D F-p modeli pri dĺžke kroku menšej ako 2.0 s. Úspešná paralelizácia 1D heliosferických modelov nás viedla k možnosti paralelizovať 2D F-T, 2D F-p a 2D B-p na GPU. Napriek vyššej pamäťovej náročnosti sa dosiahlo ešte vyššie zrýchlenie ako pri 1D modeloch pri zachovaní prijateľnej presnosti. GPU implementácia 2D F-T modelu vykázala 9.83-násobné zrýchlenie oproti referenčnému multiprocesovému systému. Priemerné zrýchlenie pre 2D F-p model bolo 11.69-násobné oproti referenčnému multiprocesovému systému. GPU implementácia 2D B-p modelu vykazovala zrýchlenie medzi 82.29 až 473.46 oproti referenčnému multiprocesovému systému. Na základe výsledkov z hľadiska zrýchlenia a presnosti dosiahnutej v GPU implementáciach modelov sme vytvorili distribuovaný systém. Distribuovaný systém bol navrhnutý a implementovaný tak, aby distribuoval jednotlivé časti simulácii medzi jednotlivé uzly s rôznymi grafickými kartami. Na distribuovanom systéme s dvoma rôznymi grafickými kartami bolo dosiahnuté o 24.04% až 34.62% väčšie zrýchlenie oproti referenčnej GPU.

| | |
|---|---|
| Thesis title: | Paralelization of two-dimensional models of cosmic rays distribution in heliosphere |
| Department: | Department of Computers and Informatics, Technical University of Košice |
| Author: | Bc. Michal Solanik |
| Supervisor: | RNDr. Pavol Bobík PhD. |
| Tutor: | doc. Ing. Ján Genči PhD. |
| Date: | 4. 5. 2020 |
| Keywords: | CUDA, GPGPU, distributed systems, heliosphere |

Abstract: The goal of the previous bachelor thesis was to parallelize 1D models of cosmic rays distribution in the heliosphere. During testing 1D F-p model we discovered an error that we've named pulsation. We suggested three hypotheses that probably could cause pulsation, but none of them was confirmed. After analysis of the range of input parameters, we found out that pulsation is present in 1D F-p simulation with step length less than 2.0 s. Successful parallelization of 1D heliosphere models led us to the option to parallelize 2D B-p, 2D F-p, and 2D F-T models on GPU. Despite higher consumption of memory even higher speed up was achieved than in the case of 1D models with acceptable accuracy. GPU implementation of the 2D F-T model achieved 9.83 speed up against the reference multiprocessor system. The average speedup for the 2D F-p model was 11.69 against the reference multiprocessor system. GPU implementation of the 2D B-p model speed up ranged from 82.29 to 473.46 against the reference multiprocessor system. Positive results in terms of speed up and acceptable accuracy led us to design and implement a distributed system. Design and implementation took into account that every single node can have different GPUs. It was reflected in the distribution of simulation iterations for every node of the distributed system. On a distributed system with two different GPUs was achieved speed up that ranged from 24.04% to 34.62% against single reference GPU.

57708

**TECHNICKÁ UNIVERZITA V KOŠICIACH**

FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Katedra počítačov a informatiky

# ZADANIE DIPLOMOVEJ PRÁCE

Študijný odbor: **Informatika**
Študijný program: **Informatika**

Názov práce:

**Paralelizácia dvojrozmerných modelov modulácie kozmického žiarenia v heliosfére**

Parallelization of two-dimensional models of cosmic ray modulation in the heliosphere

Študent: **Bc. Michal Solanik**
Školiteľ: **RNDr. Pavol Bobik, PhD.**
Školiace pracovisko: **Oddelenie kozmickej fyziky, Ústav experimentálnej fyziky SAV Košice**
Konzultant práce: **doc. Ing. Ján Genči, PhD.**
Pracovisko konzultanta: **Katedra počítačov a informatiky**

Pokyny na vypracovanie diplomovej práce:

1. Analyzovať 2D model simulácii distribúcie kozmického žiarenia v heliosfére bez driftových procesov a určiť vhodné platformy pre paralelizáciu.
2. Vysvetliť pôvod chyby (pulzácie vo vypočítanom energetickom spektre kozmického žiarenia na 1AU) pri GPU implementácii 1D F-p modelu pri malých časových krokoch.
3. Implementovať 2D modely s Parkerovým a modifikovaným Parkerovým heliosférickým magnetickým poľom na zvolených platformách.
4. Otestovať presnosť a zrýchlenie 2D modelov oproti referenčnému multiprocesorovému systému.
5. Vypracovať dokumentáciu podľa pokynov vedúceho práce.

Jazyk, v ktorom sa práca vypracuje: slovenský
Termín pre odovzdanie práce: 04.05.2020
Dátum zadania diplomovej práce: 31.10.2019

prof. Ing. Liberios Vokorokos, PhD.
dekan fakulty

**Čestné vyhlásenie**

Vyhlasujem, že som záverečnú prácu vypracoval(a) samostatne s použitím uvedenej odbornej literatúry.

Košice, 4.5.2020 .....

*Vlastnoručný podpis*

**Poďakovanie**

Na tomto mieste by som rád poďakoval svojmu vedúcemu práce, RNDr. Pavlovi Bobíkovi PhD., za jeho čas a odborné vedenie počas riešenia mojej záverečnej práce a vývoj dvojrozmerných modelov propagácie častíc kozmického žiarenia v heliosfére pre túto záverečnú prácu.

Rovnako by som sa rád poďakoval doc. Ing. Jánovi Genčimu PhD. za za cenné rady.

# Obsah

# Zoznam obrázkov

# Zoznam tabuliek

# Úvod

V bakalárskej práci sme sa zaoberali [1] paralelizáciou 1D F-p a B-p modelov propagácie kozmického žiarenia heliosférou na GPU firmy Nvidia pomocou rámca CUDA. V prípade 1D F-p modelu sme dosiahli 7.87-násobné zrýchlenie oproti referenčnému multiprocesorovému systému. V prípade 1D B-p modelu sme dosiahli najmenej 86.87-násobné zrýchlenie. To prirodzene viedlo k myšlienke, že paralelizácia na GPU by mohla pomôcť zrýchliť aj dvojrozmerné modely pre určovanie distribúcie kozmického žiarenia v heliosfére. So zložitosťou fyzikálnych modelov rastie aj ich dopyt po výkone.

Paralelizácia dvojrozmerných modelov prináša ďalšie problémy v podobe komplexnejších výpočtov, s väčším počtom vstupných parametrov, ktoré bude nutné optimalizovať pre dané platformy, na ktorých sa výpočty budú realizovať.

V prípade paralelizácie na GPU bude nutné zvážiť a otestovať, či implementácia daných dvojrozmerných modelov je paralelizovateľná na GPU. Limitácia množstvom rýchlej pamäte môže znamenať taktiež to, že implementácia pre multiprocesorové systémy bude výrazne efektívnejšia, ako v prípade implementácie pre GPU.

Pri používaní GPU implementácie 1D F-p modelu v nasledujúcich mesiacoch po odovzdaní bakalárskej práce bolo zistené, že v prípade dĺžky kroku dt = 0.5 sekundy, sa v spektre, z GPU implementácie 1D B-p modelu, nachádzajú pulzy, ktoré by sa v hladkom priebehu funkcie spektra nemali vyskytovať. Pri testovaní [1] hraničných hodnôt difúzneho koeficientu K a rýchlosti slnečného vetra V s dĺžkou kroku dt = 5.0 s sme pulzy nezaznamenali. Preto usudzujeme, že tieto pulzy narastajú so znižujúcim sa časovým krokom výpočtu. Analýza a hľadanie príčin tejto chyby nám poskytne ďalší náhľad na limity, ktoré sa nám vyskytli pri GPU implementácii 1D F-p modelu.

Cieľom tejto práce je paralelizovanie dvojrozmerných heliosferických mode-

lov, zrýchlenie času ich vykonávania a taktiež zvýšenie produktivitu práce tým, že výsledky z týchto modelov budú získané rýchlejšie, ako v prípade výpočtov realizovaných na multiprocesorových systémoch. Následne tieto modely implementujeme [1] do používateľského rozhrania, ktoré bolo vytvorené pre jednorozmerné heliosferické modely.

Pri riešení práce sa ukázalo, že vhodným riešením pre implementáciu dvojrozmerných heliosferických modelov bude vytvorenie distribuovaného systému, ktorý bude využívať výpočtovú silu GPU.

# 1 Formulácia úlohy

Hlavnými cieľmi tejto práce je paralelizácia dvojrozmerných heliosferických modelov - F-p, B-p, F-T modelov bez driftu a analýza zistených rozdielov, ktoré sa vyskytli v podobe pulzácie vo výsledných spektrách vypočítaných pomocou GPU implementácie 1D F-p modelu. Pulzácie neboli pri tom zaznamenané vo výsledných spektrách vypočítaných pomocou CPU implementácie 1D F-p modelu. Tieto hlavné ciele môžu byť rozdeliť na nasledujúce kroky:

- Analýza zistených rozdielov medzi CPU a GPU implementáciou 1D F-p modelu. Je potrebné ohraničiť rozsah dlžok časového kroku dt, pri ktorých sa vyskytuje, resp. nevyskytuje pulzácia. Následne bude vykonané testovanie pre výskyt pulzácie aj v dvojrozmerných modeloch.
- Paralelizácia dvojrozmerných heliosferických modelov. Vzhľadom na úspešnosť prepisu [1] jednorozmerných heliosferických modelov s ohľadom na dosiahnuté výsledné zrýchlenie bude zvolená GPU ako prvá platformu. Ak paralelizácia na GPU nebude prinášať uspokojivé zrýchlenie oproti multiprocesorovým systémom, riešenie bude potrebné realizovať na multiprocesorových systémoch.
- Optimalizácia výsledných implementácii. Výsledné implementácie bude potrebné optimalizovať na zvolené platformy. Optimalizácia bude vykonávaná najmä s ohľadom na zrýchlenie daných výpočtov.
- Analýza presnosti a získaného zrýchlenia. Z hľadiska presnosti bude potrebné zachovať presnosť čo najviac podobnú presnosti, ktorú dosahujú implementácie určené pre CPU. Bude navrhnuté testovanie, pomocou ktorého bude možné overiť presnosť jednotlivých modelov. Budeme analyzovať aj získané zrýchlenie pri jednotlivých modeloch voči referenčnému multiprocesorovému systému.

Výsledné implementácie dvojrozmerných heliosferických modelov budú implementované do používateľského rozhrania v príkazovom riadku, ktoré sme vytvorili [1] pre jednorozmerné heliosferické modely.

# 2 Pulzy v 1D F-p modeli

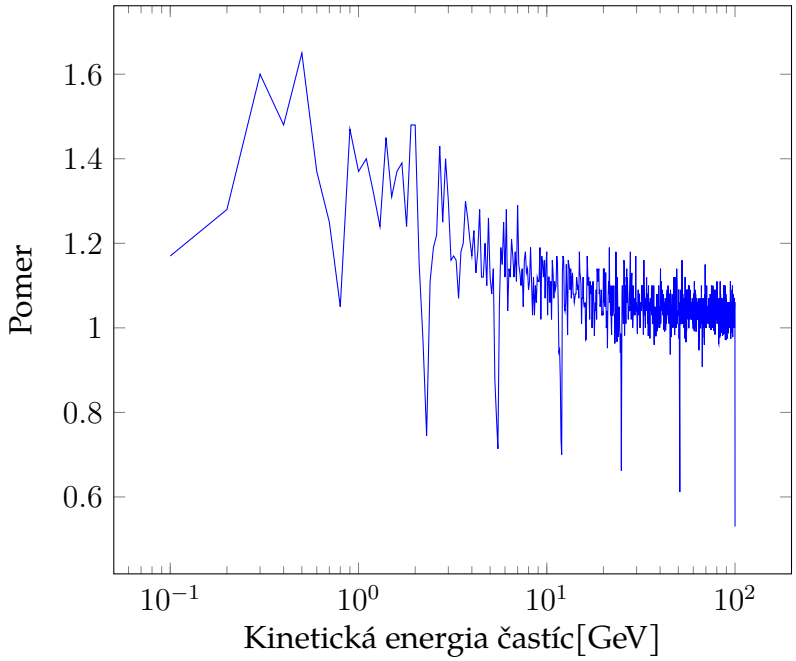
Pri dlhšom používaní 1D F-p modelu na GPU sa pri výsledných spektrách ukázal nasledujúci problém. Pri preskoku častice cez registračnú hranicu, pri dostatočne dlhom časovom kroku, častica v niektorých prípadoch preskočila registračnú hranicu až do vzdialenosti bodu zrkadlenia. Kód v takom prípade vrátil časticu na predchádzajúce súradnice a krok zopakoval nanovo. Toto opakoval, až pokiaľ nenašiel takú hodnotu kroku, ktorá skončila ešte pred zrkadlením. To viedlo k dvom problémom.

Prvým problémom s minoritným vplyvom na presnosť výsledkov bolo umiestnenie podmienky zrkadlenia v kóde. V pôvodnej verzii kódu bola podmienka zrkadlenia umiestnená za registráciou častíc, čo spôsobovalo, že častica mohla byť zaregistrovaná viacnásobne. Tento problém bol odstránený presunutím zrkadlenia ešte pred registráciu samotných hodnôt častice.

Druhým významnejším problémom bolo to, že pri dlhom kroku (od registračnej hranice po zrkadlenie) kód v podstate robil selekciu krokov vypočítaných z malých náhodných čísel. Použitá distribúcia náhodných čísel tak nebola správna. Pre riešenie situácie sme navrhli jednoduché zníženie dĺžky časového kroku. Pri dostatočne nízkom časovom kroku sa nebude stávať, že by dĺžka kroku bola väčšia ako vzdialenosť medzi bodom zrkadlenia a registračnou hranicou. Po aplikácii krátkeho kroku, ktorý viedol k predlženiu výpočtu, sa ukázali dve nové chyby 1D riešenia pre GPU. Prvým bol problém s tvarom spektra pri veľmi krátkom kroku. Spektrum v takom prípade vykazovalo dvojicu nových zjavne nekorektných charakteristík. Prvou bola vyššia intenzita pre nižšie energie, získaná z F-p výpočtu. Druhou boli pulzy, vyskytujúce sa v spektre cez celý jeho rozsah.

Pri testovaní zmeny pozície zrkadlenia a skrátení časového kroku na $dt = 0.5$ sme vo výslednom spektre zaznamenali nepravidelné pulzy, ktoré sú dobre viditeľné v grafe 2.1.

Obr. 2.1: Energetické spektrum pre 200 miliárd častíc - F-p model - dt=0.5




Identifikovali sme niekoľko možností, ktoré by mohli spôsobovať problém, resp. problémy, keďže nevieme určiť, či pulzy boli spôsobené jednou alebo kombináciou viacerých možností:

- Problémy spojené s distribúciou čísel v použitom generátore náhodných čísel
- Rozdiely medzi dvojitou a jednoduchou presnosťou
- Použitie optimalizačných prepínačov pri kompilácii
- Nesprávna distribúcia počtu simulácii na GPU

V riešení pre 1D F-p model je použitý generátor náhodných čísel XORWOW. Vykonali sme testovanie [1] generátora náhodných čísel XORWOW na implementácii algoritmu náhodného kráčania pri štatistike 100 miliónov náhodných čísel. Z hľadiska presnosti sme testovali korektnosť distribúcie náhodných čísel, počet preskokov a konvergenciu strednej hodnoty. Distribúcia náhodných čísel zo simulácie zodpovedala očakávanej distribúcii, histogram počtu preskokov zodpovedal predpokladom. Konvergencia strednej hodnoty k analytickej strednej hodnote s

maximálnou odchýlkou 0.2% bola dosiahnutá už pri štatistike $10^5$ náhodných kráčaní. Z týchto testovaní na náhodnom kráčaní sme nezistili významné odchýlky od teoretickej distribúcie náhodných čísel.

V oficiálnej dokumentácii ku knižnici cuRAND [2] sú uvedené výsledky z testov z rámca TestU01[1]. Tie uvádzajú, že generátor náhodných čísel XORWOW prešiel všetkými testami, ale pri niektorých sa objavila štatistika, ktorá nezodpovedala úplne požadovanej distribúcii[2].

Kvôli architektúre a použitiu [4] GPU nie je na väčšine GPU podporovaná dvojitá presnosť natívne, ale len softvérovo. To nás viedlo [1] k zmene presnosti z dvojitej na jednoduchú v prepisovanom kóde. Takáto zmena vedie ku kumulovaným nepresnostiam, ktoré sa prejavia najmä pri dlhšie trvajúcich simuláciach.

Pri kompilácii prepísaného kódu využívame [1] prepínač *–use_fast_math*, ktorý umožňuje využívať hardvérové prostriedky GPU na výpočty. Tento prepínač má za dôsledok aktiváciu [4] troch ďalších optimalizačných prepínačov a to:

- *–ftz=true* - pri povolení tohto parametra sú čísla v denormalizovanej forme nahradené nulami. Harris poukázal [5] na to, že po skompilovaní kódu, ktorý vykonáva jednoduché číselné operácie, verzia, pri ktorej bol parameter *–ftz* zapnutý, bola až o 20% rýchlejšia.

- *–prec-div=false* - povolí aproximáciu výsledku namiesto zaokrúhľovania[3] v prípade delenia.

- *–prec-sqrt=false* - povolí aproximáciu výsledku namiesto zaokrúhľovania[4] v prípade odmocnín.

- *–fmad=true* - zmenšenie presnosti čísel v pohyblivej rádovej čiarke v prípade sčítania, odčítania a násobenia.

Pri prepise riešenie pre 1D F-p model sme museli [1] zmeniť spôsob distribúcie počtu simulácii kvôli inému výpočtovému modelu používanému na GPU. Pri dlhších časových krokoch $dt > 0.5$, ktoré sme testovali, sa neobjavili žiadne výkyvy.

[1] Rámec určený [3] na testovanie generátorov náhodných čísel

[2] Dokumentácia [2] neuvádza závažnosť týchto chýb a ich ďalšie potencionálne vplyvy pri vyššej štatistike

[3] Zaokrúhľovanie prebieha [4] podľa IEEE 754

[4] Zaokrúhľovanie prebieha [4] podľa IEEE 754

Preto predpokladáme, že pri dlhšie trvajúcich simuláciach, ktoré sú presnejšie, to môže spôsobovať problémy z hľadiska štatistiky.

# 3 Paralelizácia 2D heliosferických modelov

Našim cieľom v tejto kapitole je analyzovať možnosti paralelizácie, navrhnúť optimalizácie, ktoré je možné vykonať na danom dvojrozmernom riešení. Budeme sa venovať aj analýza možných chýb, ktoré sa ukázali pri používaní jednorozmerného riešenia F-p modelu.

## 3.1 Paralelné počítačové systémy

Paralelné počítačové systémy poskytujú [6] oproti sériovým počítačom výhody v podobe kratšieho vykonávacieho času, možnosti vykonávať viacero rôznych separátnych úloh naraz alebo možnosti riešiť rozsiahlejší problém, ak sa jedná o systémy pracujúce v reálnom čase.

Simulácie propagácie častíc kozmického žiarenia v heliosfére sú náročné na výpočtový výkon. V prípade implementácie 2D F-T modelu sme na referenčnom multiprocesorovom systéme namerali vykonávací čas 150 hodín pre časový krok dt = 5.0 pre 100 miliárd simulácii pri behu na 64 jadrách CPU. Pre prípadné urýchlenie výpočtu by sme mohli zvýšiť časový krok dt, problémom je ale to, že pri 1D aj 2D modeloch kozmického žiarenia v heliosfére sa vyskytuje [7] chyba, ktorá mení tvar spektra pri vyšších energiách ako približne 100 GeV. Preto našu snahu zameráme na zrýchlenie - zefektívnenie vykonávacieho času vo forme hľadania iných architektúr, pomocou ktorých by sme vedeli dosiahnuť zrýchlenie oproti referenčnému multiprocesorovému systému.

Z hľadiska flynnovej taxonómie môžeme paralelné počítačové systémy rozdeliť [8] na:

- SISD - jeden inštrukčný prúd, jeden dátový prúd(angl. Single Instruction

stream, Single Data stream),

- SIMD - jeden inštrukčný prúd, viacero dátových prúdov(angl. Single Instruction stream, Multiple Data stream),
- MISD - viacero inštrukčných prúdov, jeden dátový prúd(angl. Multiple Instruction stream, Single Data stream),
- MIMD - viacero inštrukčných prúdov, viacero dátových prúdov(angl. Multiple Instruction stream, Multiple Data stream).

Pre paralelizáciu simulácii kozmického žiarenia v heliosfére sú ideálne systémy typu SIMD a MIMD. Kým v prípade systémov tpyu SISD sa jedná o sekvenčný počítač, tak v prípade MISD sa jedná o viacero operácii nad rovnakými dátami, čo nie je prípad simulácii propagácie kozmického žiarenia v heliosfére.

Z hľadiska programovacieho modelu, ktorý sa používa [4] v rámci CUDA môžeme zaradiť grafické karty, vektorové procesory medzi systémy typu SIMD. V prípade grafických kariet sa jedná presnejšie o model typu SIMT - jeden inštrukčný prúd, viacero vlákien(angl. Single Instruction stream, Mutliple Threads), ktorý využíva v rámci jednotlivých procesorov aj viacero vlákien. Systému typu SIMD sú pre nás výhodné z toho hľadiska, že jednotlivé simulácie majú iné vstupné parametre odvíjajúce sa od počiatočnej energie $T_{kin}$, pri čom rovnaký typ simulácie chceme vykonať nad celou množinou vstupných údajov.

Podľa Tanenbauma a spol. medzi systémy typu MIMD môžeme radiť [9] multiprocesory a multipočítače. Kým multiprocesory využívajú na medziprocesorovú komunikáciu zdieľanú pamäť, tak multipočítače využívajú na komunikáciu medzi jednotlivými uzlami správy. Z hľadiska simulácii kozmického žiarenia v heliosfére vieme usúdiť, že momentálne používané multiprocesorové systémy neposkytujú dostatočný výkon pre získanie potrebnej štatistiky.

Z hľadiska výkonu by ale možným riešením mohli byť multipočítače - distribuované systémy. Distribuované systémy definuje [10] Tanebuam a spol. ako skupinu nezávislých počítačov, ktoré sa z hľadiska používateľa javia ako jeden systém. Autori radia medzi základné vlastnosti distribuovaného systému to, že komunikácia medzi jednotlivými počítačmi nie je viditeľná pre používateľa daného systému takisto, ako vnútorná štruktúra distribovaného systému. Taktiež radia ľahkú škálovateľnosť systému medzi základné vlastnosti. Práve možnosť ľahkej škálovateľnosti je pre simulácie modelov kozmického žiarenia v heliosfére výhodná.

Využitie viacerých multiprocesorových systémov, respektíve využitie viacerých systémov, ktoré využívajú výpočtovú silu GPU, je vhodnou cestou pre získanie potrebného zrýchlenia oproti výpočtom realizovaných na referenčnom multiprosorovom systéme.

## 3.2 Paralelizácia 2D modelov na úrovni GPU

V prípade 1D modelov[1] sme dokázali [1], že prepis a paralelizácia pre GPU boli úspešné. Pri GPU implementácii 1D F-p modelu sme dosiahli 7.87-násobné zrýchlenie oproti referenčnému multiprocesorovému systému. V prípade 1D B-p modelu sme dosiahli najmenej 86.87-násobné zrýchlenie. Pre tieto modely bolo následne implementované používateľské rozhranie v príkazovom riadku, cez ktoré bolo možné spúšťať a zadávať konkrétne parametre pre jednotlivé modely.

Pri 2D modeloch tento prepis už nemusí byť taký efektívny[2], ako pri 1D modeloch z toho dôvodu, že tieto modely sú komplexnejšie, čo sa týka vstupných parametrov a taktiež samotný výpočet obsahuje viac rovníc väčšej komplexity ako pri modeloch v 1D.

$$\frac{\partial f}{\partial t} = \frac{1}{r^2}\frac{\partial}{\partial r}(r^2 k_{diff}\frac{\partial}{\partial r}f) - \frac{1}{r^2}\frac{\partial r^2 V_{SW} f}{\partial r} + \frac{1}{3}(\frac{1}{r^2}\frac{\partial r^2 V_{sw}}{\partial r})\frac{1}{p^2}\frac{\partial}{\partial p}(p^3 f) \tag{3.1}$$

$$\begin{aligned}\frac{\partial f}{\partial t} = \frac{1}{r^2}\frac{\partial}{\partial r}(r^2 K_{rr}\frac{\partial f}{\partial r}) + \frac{1}{r^2\sin(\theta)}\frac{\partial}{\partial \theta}(K_{\theta\theta}\sin(\theta)\frac{\partial f}{\partial \theta}) \\ + \frac{1}{3r^2}\frac{\partial}{\partial r}(r^2 V)\frac{\partial}{\partial T}(\Gamma T f) - \frac{1}{r^2}\frac{\partial}{\partial r}(r^2 V f)\end{aligned} \tag{3.2}$$

Porovnajme riešenie [11] Fokker-Planckovej rovnice pre 1D F-p model bez driftu uvedené v rovnici 3.1 s riešením [12] Fokker-Planckovej rovnice pre 2D F-T model bez driftu uvedenej v rovnici 3.2. Z porovnania riešení oboch rovníc vieme usúdiť nasledujúce:

- Vyššia pamäťová náročnosť - môžeme predpokladať, že pamäťová náročnosť pôjde nahor. Vyplýva to z väčšieho počtu členov v rovnici pre 2D F-T model. Kým pri implementácii 1D F-p modelu sme dosiahli [1] 75% využitie

[1] 1D F-p a B-p model

[2] Pri 1D modeloch bolo preukázané [1] využitie GPU na 75%

výpočtovej kapacity referenčnej GPU[3], pri implementácii 2D F-T modelu využiteľnosť GPU môže klesnúť pod hranicu, kedy rýchlosť implementácie na GPU je citeľne pomalšia oproti CPU verzii. Optimalizáciu z pamäťového hľadiska musíme z časti prenechať na kompilátor. Vieme obmedziť [4] maximálny počet registrov parametrom -maxrregcount, ktoré môžu byť použité pri vykonávaní funkcií na GPU, aby využiteľnosť GPU neklesla kvôli presahu zopár registrov. To spôsobí vynútenie si používania pomalších pamätí pre premenné, ktoré sa nezmestia do registrov.

- Vyššia výpočtová náročnosť - usudzujeme, že pri vyššom počte členov a matematických operácii v rovnici pre 2D F-T model oproti rovnici pre 1D F-p model bude viesť k vyššej výpočtovej náročnosti. Ďalšími problémami, ktoré môžu vyplývať z vyššej výpočtovej náročnosti, sú problémy s presnosťou. Vykonali sme [1] test, pri ktorom sme sledovali vybranné premenné. Po 500 tisíc iteráciach bola najvyššia odchýlka pri vstupnej kinetickej energie $T_{kin} = 0.0001$ a dĺžke kroku $dt = 5.0$ na úrovni 4.28% oproti GPU verzii. Po 500 tisíc iteráciach pre vstupnú kinetickú energiu $T_{kin} = 20.0001$ a dĺžku kroku $dt = 5.0$ sme zaznamenali odchýlku na úrovni maximálne 0.61% oproti GPU verzii. Taktiež nesmieme zabudnúť na pulzy pri implementácii 1D F-p modelu na GPU spomínané v kapitole 2, ktoré sa môžu objaviť aj pri implementácii 2D F-T modelu.

- Paralelizovateľnosť simulácii - už pri implementácii 1D F-p alebo B-p modelu sme preukázali [1], že paralelizácia simulácii na menšie časti nie je optimálnym riešením. Ak by sme vykonávali simulácie s určitým počtom krokov, museli by sme tieto simulácie opakovane púšťať, ukladať si ich priebežný stav, čo by vyústilo k problémom nielen v podobe vyššej pamäťovej náročnosti, ale značne by sa zvýšil podiel komunikačnej réžie GPU kvôli opakovanému spúšťaniu simulácii.

[3] GTX 1080TI

## 3.3 Vlastnosti distribuovaných systémov a ich vplyv na koncepciu

V kapitole 3.1 sme posudzovali vhodnosť vytvorenia distribuovaného systému, ktorý bude mať na starosti výpočet simulácii kozmického žiarenia v heliosfére. V kapitole 3.2 sme analyzovali možnosť využitia výpočtovej sily GPU na výpočty. Ak by sme spojili distribuovaný systém spolu s využitím realizovania výpočtov na GPU, získali by sme oveľa väčšiu výpočtovú silu než v prípade spojenia rovnakého počtu multiprocesorových systémov.

Základné ciele distribuovaných systémov môžeme rozdeliť [10] na:

- dostupnosť zdrojov - jedným z cieľov distribuovaných systémov je uľahčiť prístup používateľov k vzdialeným, poprípade väčšiemu množstvo zdrojov, ktoré má používateľ k dispozícii,
- transparentnosť - ide o vlasnosť systému skrývať svoju vnútornú štruktúru pred používateľom systému,
- otvorenosť - dodržiavanie rozhraní služieb a štandardov komunikácie,
- škálovateľnosť - vlastnosť rozširovania systému,
- tolerancia chýb - distribuovaný systém by sa mal sám zregenerovať z chýb, ktoré môžu nastať.

Z hľadiska distribuovaného systému, ktorý by mal na starosti simulácie kozmického žiarenia je dôležitá najmä dostupnosť výpočtových zdrojov, škálovateľnosť a tolerancia chýb. Autori Tanenbaum a spol. radia medzi zdroje [10] v rámci distribuovaného systému napríklad počítače, úložný priestor, súbory. Možnosť pustenia simulácie kozmického žiarenia v heliosfére na viacerých dostupných systémoch šetrí čas nielen z hľadiska najdlhšieho vykonávacieho času (predpokladajme, že systém by rozdeľoval simulácie podľa výkonu jednotlivých jednotlivých uzlov alebo dynamicky podľa potreby), ale aj z hľadiska času, ktorý je potrebný na určenie spektra z výpočtov zo všetkých systémov, na ktorých dané simulácie boli vykonávané.

Na problémy plynúce zo škálovateľnosti sa môžeme [10] pozerať z hľadiska centralizácie služieb, údajov a algoritmov. Z hľadiska služieb sa môže jednať o

centralizovaný server, čo podľa autorov Tanenbauma a spol. je vždy problém, ktorému sa nie vždy dá vyhnúť. Problémy škálovateľnosti pri centralizovaní údajov môžu predstavovať problém kvôli potrebe získavania informácii v rámci systému z jedného miesta. Centralizácia algoritmov môže viesť k veľkému množstvo správ medzi jednotlivými uzlami distribuovaného systému, čo môže mať za následok preťaženie siete. Škálovateľnosť pre distribuovaný systém so simuláciami kozmického žiarenia v heliosfére by bola jedným z kľúčových prvkov. Jednotlivé heliosferické modely sú postupom času vyvíjané, rozširované. Okrem narástu komplexnosti výpočtov na strane simulácii budeme taktiež chcieť zvyšovať výkon distribuovaného systému. Vďaka nezávislosti simulácii vieme, že problémy týkajúce sa škálovateľnosti údajov a algoritmov nevzniknú.

Tolerancia chýb pokrýva [13] podľa autorov Kopetza a spol. taktiež dostupnosť, spoľahlivosť, bezpečnosť a udržiavateľnosť distribuovaného systému. Autori Tanenbaum a spol. definujú [10] schopnosť systému mať toleranciu voči chybám ako jednu zo základných vlastností. Z hľadiska distribuovaného systému so simuláciami kozmického žiarenia v heliosfére bude nutné navrhnúť mechanizmus, ktorý by zabezpečil, že jednotlivé simulácie, ktoré budú určené pre jednotlivé uzly distribuovaného systému boli monitorované a v prípade problému - neúspešného ukončenia simulácie alebo nedostupnosti uzla, sa výpočet vykoná na inom uzle.

## 3.4 Komunikácia v distribuovaných systémoch

Niektorí autori tvrdia, že distribované systémy môžeme definovať ako systémy, ktorých komunikácia medzi jednotlivými uzlami sa realizuje pomocou správ. Autori Tanenbaum a spol. rozdeľujú [10] komunikačné modely využívané v distribuovaných systémoch na tri skupiny - RPC (angl. Remote Procedure Call), MOM - stredná vrstva zameraná na správy(angl. Message-Oriented Middleware) a prúd údajov (angl. data streaming).

RPC je vzdialené [10] volanie funkcie(angl. Remote procedure call), ktoré predstavuje model, ktorý je založený na volaní funkcii na vzdialenom zariadení. MOM predstavuje takzvanú strednú vrstvu v architektúre, ktorá slúži na komunikáciu pomocou správ. Umožňuje dosiahnuť menšiu vzájomnú prepojenosť jednotlivých systémov. rúdovo orientovaná komunikácia je určená najmä pre systémy, pri ktorých je nutné prenášať veľké množstvo údajov ako napríklad multimediálne sú-

bory.

Vzhľadom k charakteristike distribuovaného systému určeného na výpočty kozmického žiarenia v heliosfére je výhodnejšie použiť MOM v prípade použitia uzlov, na ktorých sa budú realizovať výpočty na GPU. Z hľadiska spoľahlivosti a odolnosti systému voči chybám by bolo zabezpečenie fronty, do ktorej sú posielané výsledky, voči strate údajov dobrým krokom. Taktiež autori Curry a spol. tvrdia pri porovnávaní [14] MOM a RPC to, že s MOM vieme zabezpečiť vyššiu spoľahlivosť systému. Uvádzajú taktiež, že výhodou RPC je zabezpečenie sekvenčnosti výpočtu.

Medzi protokoly, ktoré sa využívajú na komunikáciu v MOM patrí [15] [16] AMQP, STOMP, MQTT. AMQP(angl. Advanced Message Queueing Protocol) je štandard, ktorý slúži na prenos správ medzi aplikaciami. Medzi charakteristické vlastnosti patrí podpora frônt, smerovania, spoľahlivosť a bezpečnosť. Zároveň podporuje komunikáciu pomocou RPC a taktiež vzor publish-subscribe. MQTT(angl. Message Queue Telemetry Transport) je binárny protokol, ktorý využíva vzor publish-subscibe. Je orientovaný najmä na IoT zariadenia, keďže je pomerne nenáročný na zdroje. STOMP(angl. Simple Text Oriented Messaging Protocol) je textovo orientovaný protokol, ktorý podporuje vzor publish-subscribe.

## 3.5 Tolerancia chýb

Jednou z vlastností, ktoré sme si spomínali v kapitole 3.3 je aj tolerancia chýb a regenerácia distribuovaného systému z týchto chýb. Autori Tanenbaum a spol. definujú [10] chyby, ktoré môžeme radiť do niekoľkých kategórii:

- chyby vyplývajúce zo zlyhania systému, kde patria chyby, kedy sa vykonávanie na uzle distribuovaného systému zasekne,
- chyby vyplývajúce z nesprávnej odpovede, kde odpoveď uzla distribuovaného systému je nesprávna,
- chyby vyplývajúce z vynechania kroku pri komunikácii, kedy uzol distribuovaného systému neobdrží prichádzajúce správy alebo zlyhá pri odosielaní správ,
- chyby vyplývajúce z časovania, kedy uzol distribuovaného systému neodpovie v požadovanom časovom intervale,

- svojvoľné zlyhania, medzi ktoré zaradzujeme nesprávny výstup, ktorý nemôže byť rozpoznaný ako nesprávny.

Vymenované chyby môžu spôsobiť nesprávne fungovanie distribuovaného systému, resp. ohroziť jeho celkovú funkcionalitu. Autori Tanenbaum a spol. tvrdia [10], že na to, aby bol systém odolný voči chybám, tak musí skrývať stav zlyhania voči ostatným uzlom v distribuovanom systéme. Jedným zo spôsobov, ako to dosiahnuť je redundancia. Podľa autorov je ju možné v prípade distribuovaných systémov rozdeliť na informačnú, časovú a fyzickú redundanciu. Zároveň redundanciu označujú za kľúčový prvok v prípade odolnosti voči chybám.

Okrem redundancie autori Tanenbaum a spol. pomenúvavajú [10] spoľahlivú komunikáciu v skupinách, obnovenie po chybe ako ďalšie dôležité aspekty v prípade odolnosti distribuovaného systému voči chybám.

## 3.6 Distribuované GPU systémy vo vedeckej oblasti

Výkon GPU je vysoký, ale na niektoré simulácie, výpočty, najmä v reálnom čase, nestačí výkon jednej GPU. Vo vedeckej oblasti preto vznikli systémy, ktoré sa snažia riešiť dané problémy distriubovane aj na GPU. Implementáciu riešenia 1D F-p alebo 2D F-T alebo iného 2D modelu nie je nutné simulovať v reálnom čase, ale ukázalo sa, že pri kombinácii niektorých vstupných parametrov môže aj výpočet na GPU trvať [1] rádovo niekoľko dní, poprípade až týždňov na nazhromaždenie dostatočne vysokej štatistiky. Systémy, ktoré obsahujú viacero grafických kariet, nedokážu využiť plnú výpočtovú silu všetkých zapojených GPU na 100%. Autori Castro a spol. preukázali [17], že efektivita viacerých GPU prepojených pomocou SLI v jednom systéme klesá. Distribuovaný systém tým pádom prinesie nielen väčšiu výpočtovú silu, ale taktiež ľahkú rozšíriteľnosť daného systému. Negatívom tohto prístupu je zložitejšia implementácia s ohľadom na komunikáciu medzi jednotlivými uzlami distribuovaného systému.

Autori Lončar a spol. vo svojom článku prezentujú [18] distribuovaný systém, ktorý slúži na simuláciu pohybu svalu. S pomocou OpenMP dokázali dosiahnuť na referenčnej pracovnej stanici so 16 jadrami 2 až 12-násobné zrýrchlenie, kým s verziou využívajúcou OpenMP a MPI na počítačovom clustri s 32 jadrami dokázali dosiahnuť zrýchlenie 11.5-16.5. Pri kombinácii CUDA a MPI dokázali dosiahnuť 9-10 násobné zrýchlenie na počítačovom clustri s 32 jadrami. Autori ďalej

uvádzajú, že prezentované výsledky nie sú získané z optimalizovaných verzií daných programov.

Autori Ivanović a spol. vo svojom článku opisujú [19] taktiež distribuovaný systém, ktorý mal simulovať deformácie jazyka počas prežúvania. Využili k tomu CUDA a MPI pre komunikáciu medzi jednotlivými uzlami. So 4 GPU dokázali dosiahnuť 73.33-násobné zrýchlenie oproti sekvenčnej verzii.

Myšlienku distribuovaný GPU systém mali aj autori Cuomo [20] a spol. pre výpočty optického toku. Ich distribovaný systém využíval MPI a rámec CUDA, neskôr nahradený knižnciou PETSc. PETSc je [21] knižnica, ktorá obsahuje dátové štruktúry a funkcie k vedeckým výpočtom. Využíva MPI, rámec CUDA a OpenCL. Dosiahli zrýchlenie 19.93 na 8 uzloch aj s GPU oproti 4.31 násobnému zrýchleniu s 8 uzlami bez GPU.

Distribuovaný GPU systém bol taktiež použitý pri pri Ptychografii, kde v závilosti na veľkosti vstupných dát dokázal [22] dosiahnuť zrýchlenie od 10 do $10^3$ v závilosti od veľkosti vstupných údajov. Autori tvrdia, že vykonávací čas sa skrátil z hodín na rádovo minúty. V prípade zapojenia 4 GPU NVIDIA Tesla V100 oproti jednému jadru procesora sa dosiahlo 1235-násobné zrýchlenie. Bola použitá kombinácia CUDA a MPI.

## 3.7 Distribuované systémy používajúce MPI

V kapitole 3.6 sme si uviedli niekoľko príkladov distribuovaných systémov. Všetky z nich využívali MPI na komunikáciu medzi jednotlivými uzlami.

MPI je [23] špecifikácia komunikačného rozhrania slúžiaca na paralelné vykonávanie. V špecifikácii MPI-2 pribudla podpora pre škálu rôznych zariadení, t.j. je možná paralelizácia na úrovni CPU, uzlov distribuovaného systému.

Všetky GPU systémy, ktoré sme spomínali v kapitole 3.6 boli realizované v laboratórnych podmienkach. Považujme dané laborátorne podmienky za priam ideálne, kde sú minimálne, až žiadne problémy s konektivitou. Medzi ďalší faktor môžeme radiť, že implementácie daných javov, ktoré boli simulované, boli už v dospelom stave - t.j. implementácia bola ošetrená voči drvivej väčšine stavov, ktoré môžu spôsobiť pád aplikácie. Autori ani v jednom článku nepopisujú [18] [19] [20] [22] odolnosť voči chybám, ich ošetrenie atď..

Jonathan Dursi tvrdí [24], že MPI nie je vhodný nástroj na použitie v dnešnej

dobe a to z niekoľkých dôvodov:

- Nevhodná úroveň abstrakcie - Dursi [24] považuje úroveň abstrakcie MPI za nevhodnú. Uvádza pri tom ten istý príklad riešenie 1D difúznej rovnice s použitím knižnice MPI, rámca Spark a programovacieho jazyka Chapel. Autor pripúšťa, že to nie je celkom férové porovnávanie. Spark je [25] rámec, ktorý slúži na vytváranie distribuovaných aplikácii. Chapel je [26] programovací jazyk, ktorý je priamo určený na paralené operácie na veľkých škálovateľných systémoch. Autor tým pádom porovnával rôznu úroveň abstrakcie pri prístupe ku problému, čo viedlo k tomu, že verzia s MPI má o viac ako dvojnásobný počet riadkov ako implementácia s použitím rámca Spark alebo programovacieho jazyka Chapel. Tento rozdiel je prevažne tvorený kódom, ktorý zaisťuje základnú funkcionalitu, ako inicializácia komunikácie, posielanie správ a pod.. Poskytuje to na jednu stranu výhodu toho, že vieme mať celú komunikáciu pod našou kontrolou, ale na druhú stranu, hromadí sa kód, ktorý celkovo môže zneprehľadňovať riešenie. To môže viesť [27] k problémom s čistotou kódu, ktorú je potrebné dodržiavať k rozširovaniu funkcionality a udržiavaniu existujúceho kódu.

- Problematická tolerancia chýb - autor považuje toleranciu chýb pri MPI za nedostatočnú. Uvádza, že je to spôsobené najmä snahou tvorcov o zachovanie spätnej kompatibility so staršími verziami MPI. Tieto tvrdenia potvrdzuje množstvo projektov, resp. množstvo implementácii, ktoré mali priniesť istú úroveň tolerancie chýb do MPI. Autori Hursey a spol. sa snažili implementovať odolnosť voči chybám [28] pre Open MPI na základe distribuovaných kontrolných bodov a techník reštartu. Autori Louca a spol. implementovali [29] toleranciu voči chybám pomocou Unixové skriptu, ktorý kontroluje existenciu daných procesov. Z uvedeného vyplýva, že samotná tolerancia pri chybách nie je v MPI dostatočná a je potrebná dodatočná implementácia, resp. použitie už existujúcej implementácie.

- Robustnosť MPI - autor považuje MPI za nevhodné pre úlohy, ktoré vyžadujú menej ako 128 jadier. Tvrdí, že na tejto úrovni sú vhodnejšie iné knižnice / rámce, ktoré poskytujú rovnaký výkon pre potrebné riešenie. Autor ale toto tvrdenie nepotvrdzuje ďalšími konkrétnymi príkladmi.

# 4 Návrh distribuovaného systému a testovania 1D pulzov

V tejto kapitole navrhneme postup pri testovaní a možných riešeniach chyby, resp. chýb, ktoré spôsobujú pulzy pri 1D F-p modeli. Taktiež sa budeme zaoberať návrhom jednoduchého distribuovaného systému, ktorý bude mať na starosti výpočet 1D a 2D heliosferických modelov.

## 4.1 Postup pri testovaní 1D pulzov

Vytýčili sme si 4 možné oblasti v kapitole 2, ktoré môžu spôsobovať pulzy pri 1D F-p modeli. Testovanie je vhodné vykonávať pre štatistiku prevýšujúcu 200 miliárd simulácii trajektórii, keďže táto chyba sa s dobrým odlíšením od šumu ukázala až pri takto vysokých štatistikách výpočtov s časovým krokom dt = 0.5, so štatistikou 200 miliárd simulovaných trajektórii.

Pre jednotlivé možnosti uvedené v kapitole 2 si uvedieme, akým spôsobom eliminujeme vplyv daného problému:

- Problémy spojené s distribúciou čísel v použitom generátore náhodných čísel vieme vylúčiť zámenou generátora náhodných čísel. Z testovania generátorov náhodných čísel vyplýva [2], že pseudogenerátor náhodných čísel Philox prešiel všetkými testami, pri čom nebol zaznamenaný žiadny problém so štatistikou vygenerovaných čísel. Toto nám umožní určiť alebo vylúčiť, že pulzy spôsobujú problémy s distribúciou vygenerovaných náhodných čísel psedogenerátora náhodných čísel XORWOW.

- Rozdiely medzi dvojitou a jednoduchou presnosťou nevieme otestovať v rozumnom časovom horizonte. Prvá verzia implementácie 1D F-p modelu s

dvojitou presnosťou na GPU bola [1] oproti finálnej 42.98-násobne pomalšia. Vykonavací čas implementácie 1D F-p modelu s jednoduchou presnosťou pre 200 milárd simulácii s časovým krokom dt = 0.5 bol približne 4 dni. Z tohto vyplýva, že odhadovaný vykonávací čas pre verziu implementácie algoritmu F-p modelu s dvojitou presnosťou na GPU pre časový krok dt = 0.5 a 200 miliárd simulácii by bol 5.73 mesiaca.

- Použitie optimalizačných prepínačov pri kompilácii vieme vylúčiť tak, že nepoužijeme pri kompilácii prepínač *–use_fast_math*. Ak by sa ukázalo, že pulzy sa vo výslednom energetickom spektre nenachádzajú, tak budeme musieť postpne vykonať testy aj s jednotlivými prepínačmi spomínanými v kapitole 2, aby sme vedeli zachovať čo najvyššie zrýchlenie.

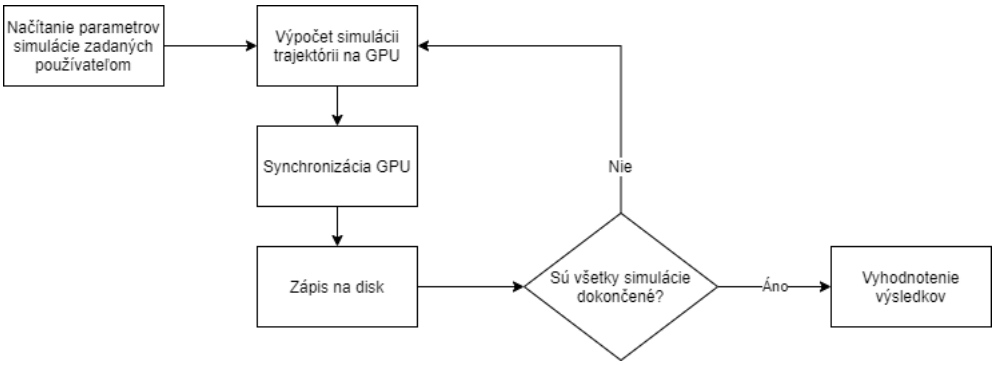
- Nesprávnu distribúciu počtu simulácii na GPU upravíme tak, aby sa čo najviac blížila jednej iterácii impleentácii algoritmu pre F-p model na CPU, t.j. 1 miliarde simulácii.

Pokiaľ by sme po jednotlivých testoch stále nevedeli prísť na to, čo spôsobuje chybu, tak budeme musieť zaznamenať podrobnosti o tejto chybe s jednotlivými vstupnými parametrami výpočtov.

## 4.2 Návrh distribuovaného systému

V kapitole 3.3 sme charakterizovali základné vlastnosti distribuovaných systémov. V kapitole 3.6 sme uviedli stručný prehľad distribuovaných systémov využívajúcich GPU vo vedeckej oblasti. Našim cieľom je navrhnúť distribuovaný systém, ktorý je určený na výpočty na úrovni 1D a 2D heliosferických modelov na GPU, zároveň bol systém od začiatku vyvíjaný tak, aby bol ľahko rozšíriteľný aj o iné fyzikálne modely.

Pre lepšie pochopenie sa musíme pozrieť na obrázok 4.1, ktorý zobrazuje principiálne spracovanie simulácii. Ak vezmeme [1] ako príklad implementácie 1D heliosférických modelov, tak po načítaní vstupných parametrov a inicializácii štruktúr generátora náhodných čísel sa začnú vykonávať výpočty na GPU. Po dokončení všetkých simulácii sa vykoná synchronizácia, kopírovanie výsledkov z pamäte GPU na pamäť RAM a následný zápis na disk. To sa deje až do vtedy, kým nie je vopred stanovený počet iterácii (vstupný parameter) dokončený.




Obr. 4.1: Realizácia [1] 1D heliosferických simulácii na GPU

Z tohto hľadiska, by využitie zdieľanej pamäte medzi jednotlivými uzlami bolo výrazne problematické. Ak by mal existovať jeden uzol, ktorý by len zapisoval a vyhodnocoval výsledky, mohlo by dôjsť pri kombinácii niektorých vstupných parametrov k zbytočnému preťaženiu siete - už len v prípade B-p modelov má každá simulácia výsledok, ktorý je zapísaný na disk. Preto vhodnejšiou metódou na prístup k riešeniu tohto problému bude nechať každý uzol samostatne vyhodnotiť výsledky svojich simulácii. To nám umožní zachovať konštantnú záťaž siete a minimalizovať ju z hľadiska údajov, ktoré predstavujú jednotlivé výsledky zo simulácii.

V kapitole 3.3 sme spomínali toleranciu chýb ako jednu zo základných vlastností distribuovaných systémov. Najdlhší realizovaný výpočet [1] v prípade optimalizovaného riešenia 1D F-p modelu na GPU trval 88.41 hodiny. V prípade, že nastane chyba, musíme očakávať od distribuovaného systému, že nájde riešenie pre tento stav a zotaví sa z chyby. Či už metódou reštartu výpočtu, alebo zvolenia iného uzla, resp. uzlov na výpočet. V kapitole 3.7 sme sa zamerali na to, čím sa vyznačuje štandard MPI a hlavne jeho negatívne dosahy. Jedným z týchto negatív bola nedostatočná tolerancia pri výskyte chýb a nevhodná abstraktná úroveň.

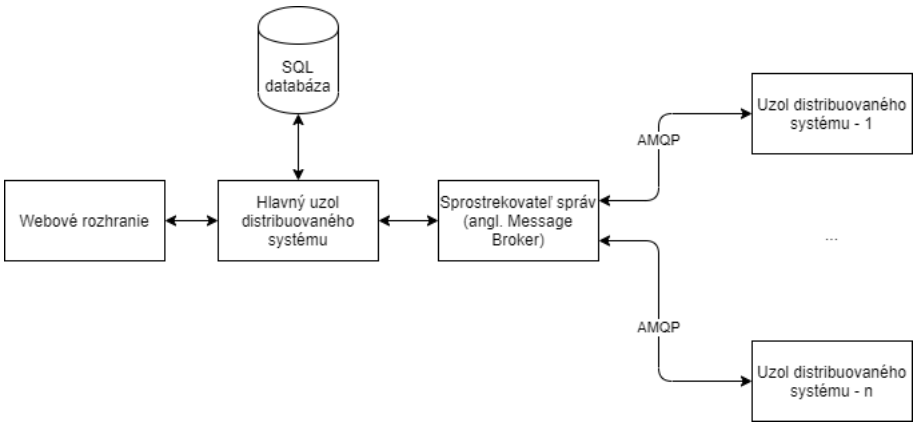
Z hľadiska GPU sme taktiež limitovaní optimalizačnými prepínačmi pri kompilácii. Bez použitia potrebných prepínačov by sme stratili značnú časť z dosiahnutého zrýchlenia, ktoré sme dosiahli obetovaním presnosti na úrovni matematických funkcií a operácii.

Stanovme si základné požiadavky na distribuovaný systém:

- systém by mal vedieť reagovať na zmenu počtu uzlov bez interakcie administrátora,

- systém by mal vedieť rozdeľovať počty simulácii tak, aby vo výsledku bol výpočet čo najrýchlejší,
- systém by mal byť odolný voči chybám - systém by sa mal zotaviť a výpočty, ktoré zlyhali, by mali byť spustené na inom dostupnom uzle,
- používateľ by mal byť schopný si vizualizovať vypočítané spektrum vo webovom rozhraní v prehliadači.

Návrh distribuovaného systému sa nachádza na obrázku 4.2. Komunikácia medzi hlavným uzlom a jednotlivými uzlami distribuovaného systému bude realizovaná cez sprostredkovateľa správ(angl. message broker) cez protokol AMQP. Vypočítané údaje z jednotlivých simulácii budú ukladané do databázy. Následne výsledky týchto jednotlivých simulácii bude možné vizualizovať vo webovom rozhraní.




Obr. 4.2: Návrh distribuovaného systému

## 4.3 Návrh hlavného uzla distribuovaného systému

V kapitole 4.2 sme navrhli základnú štruktúru distribuovaného systému. Primárnou úlohou hlavného uzla distribuovaného systému bude zabezpečenie distribúcie parametrom jednotlivým uzlom distribuovaného systému.

Z hľadiska funkcionality ako uzla distribuovaného systému vytvoríme fronty pre registráciu zariadenia, aktualizovanie stavu a pre zaznamenania výsledkov simulácii, ich zlúčenie s ostatnými výsledkami a uloženie do databázy. Registrácia

zariadenia a aktualizáciu stavu bude tvoriť jedna fronta s tým, že uzol distribuovaného systému odošle základné informácie o sebe a bude mu vrátený identifikátor, podľa ktorého vytvorí fronty, ktoré budú prislúchať danému uzlu.

Zároveň úlohou hlavného uzla bude pravidelne kontrolovať dostupnosť jednotlivých uzlov. Na jednotlivých uzloch vieme identifikovať zlyhania v rámci GPU, na druhú stranu ale nevieme identifikovať chyby, ktorých následkom je reštart počítača, problémy v elektrickej sieti alebo problémy v sieti. Kontrola sa bude realizovať na aktívnych uzloch, pri čom sa bude zároveň kontrolovať, či uzol má na starosti simulácie a či tieto simulácie prebiehajú.

## 4.4 Návrh uzlu distribuovaného systému

Úlohou uzlu distribuovaného systému bude realizovať samotné výpočty simulácii kozmického žiarenia v heliosfére. V kapitole 4.2 sme spomínali negatívny dopad MPI na abstraktnú úroveň a toleranciu chýb. Taktiež pri návrhu uzlu distribuovaného systému sme museli dbať na fakt, že optimalizačné prepínače pri kompilácii GPU implementácii modelov kozmického žiarenia v heliosfére musia byť zachované, najmä kvôli zrýchleniu, ktoré predstavujú.

Preto sme určili, že najvhodnejším rozhodnutím bude oddelenie vrstvy, ktorá bude realizovať výpočty od komunikačnej - komunikáciu bude mať na starosti ďalšia aplikácia. Proces, ktorý bude mať na starosti komunikáciu bude spúšťať optimalizovaný program pre GPU s parametrami, ktoré dorazili z komunikačnej fronty.

Po spustení aplikácie, ktorá bude mať na starosti komunikáciu prebehne registrácia v hlavnom uzle - uzol obdrží identifikátor UUID, ktorý si uloží do súboru. Zároveň s tým sa vytvoria dve komunikačné fronty - jedna, ktorá slúži na kontrolu dostupnosti (ďalej ako ping) uzla distribuovaného systému a druhá, ktorej účelom je spúšťanie simulácii. Po dokončení simulácii sa načíta výsledné spektrum zo súboru a odošle do komunikačnej fronty, ktorá slúži na zaznamenávanie výsledkov.

Nasadenie uzla distribuovaného systému môžeme realizovať cez kontajnerizáciu - Docker. Na Linuxových systémoch majú Docker kontajnery natívny prístup [30] ku výpočtovým prostriedkom GPU. Taktiež nám to umožní nasadiť uzol distribuovaného systému aj na jedno z cloudových riešení, ktoré umožňujú si prena-

jať výkon GPU.

# 5 Paralelizácia 2D modelov na úrovni GPU

Našim cieľom v tejto kapitole je popísať postup pri paralelizácii 2D F-T modelu bez driftu na GPU, popísať vzniknuté problémy pri optimalizácii a optimalizácii na architektúru Turing. Taktiež sa budeme venovať 2D F-p a B-p modelu z hľadiska implementácie a dosiahnutia optimalizácie.

Pre vývoj a overenie presnosti prepisu 2D modelov propagácie kozmického žiarenia heliosférou boli postupne vytvorené tri verzie fyzikálneho modelu. Tieto modely nezahŕňajú drift častíc v heliosfére, drift bude súčasťou verzie 4.x. Verzia 1.x je technickou verziou modelu, kvázi dvojrozmernou, pretože testuje len radiálne časti 2D modelu. Účelom tejto verzie bolo testovanie niektorých základných častí kódu, najmä z numerického hľadiska. Verzia 2.x je [31] verziou s heliosférou s Parkerovým modelom magnetického poľa. Ide o základný model poľa používaný prvé desaťročia v modeloch pre výpočet distribúcie kozmického žiarenia v heliosfére. Vhodným zdrojom informáciu o modele založenom na tomto poli je článok [32], ktorý ukazuje aj možnosť zahrnutia driftu do modelu. Verzia 3.x modelu je verziou s upraveným Parkerovým heliosférickým poľom. Toto pole je modifikované v polárnej oblasti heliosféry podľa [33]. Modifikácia zvyšuje intenzitu heliosférického magnetického poľa v polárnej oblasti. Modifikácia má za cieľ korekciu poľa k realistickejším hodnotám a v modeloch, kde je použitá, má viesť ku korekcii vysokých intenzít kozmického žiarenia prenikajúcich do vnútornej heliosféry cez polárne oblasti. Táto relatívne jednoduchá modifikácia poľa vedie k výrazne väčšej komplikovanosti difúzneho tenzora modelu, preto sú verzie 3.x komplikovanejšie voči verziám 2.x z hľadiska numerickej náročnosti modelu. Príkladom modelu s touto modifikáciou poľa je model [34].

Tabuľka 5.1: Prehľad jednotlivých verzií implementácie 2D F-T modelu

| Verzia | Obsah |
|---|---|
| 1.1 | Prechod medzi 1D a 2D F-T modelom |
| 2.0 | Prvá 2D vezia F-T modelu |
| 2.1 | Oprava chýb |
| 2.2 | Oprava chýb |
| 2.3 | Oprava chýb |
| 2.4 | Oprava chýb |
| 3.0 | Plnohodnotný 2D F-T model |
| 3.1 | Oprava chýb |
| 3.2 | Zmena injektovania |
| 3.3 | Spätná zmena injektovania, Oprava chýb |
| 3.4 | Oprava chýb |
| 3.5 | Oprava chýb |
| 3.6 | Oprava chýb |

## 5.1 2D F-T model bez driftu

Samotný prepis, resp. paralelizáciu, 2D F-T modelu sme realizovali v niekoľkých fázach. A to hlavne z toho dôvodu, že je výrazne problematické hľadať chyby, ktoré sa prejavia v spektrách relatívne zložitých priamo prepísaných modelov, vhodnejšou cestou je postupne inkrementálne pridávať do prepisu nové časti modelu a kontrolovať správnosť týchto fáz. V tabuľke 5.1 sa nachádza prehľad jednotlivých realizovaných verzií implementácie 2D F-T modelu bez driftu.

V kapitole 3.2 sme analyzovali rozdiely medzi rovnicou 1D F-p modelu a 2D F-T modelu. Pri realizácii sa musíme pozerať na počty registrov, ktoré by mali byť použité pri spúšťaní jednotlivých funkcií na GPU. Budeme využívať princípy, ktoré sme určili [1] pri implementácii 1D F-p a 1D B-p modelov - použitie funkcie atomickej inkrementácie nad premennou, ktorá značí posledné voľné miesto v poli výsledkov, výpočty jednotlivých simulácii od začiatku až do konca na GPU bez prerušovania.

Výpočty jednotlivých simulácii od začiatku do konca na GPU nám ale pri realizácii začali robiť problémy. Nejednalo sa o problémy s realizáciou, ale pamäťo-

Tabuľka 5.2: Využitie registrov implementácie 2D F-T modelu bez driftu na GPU pre architektúru Pascal

| Verzia | Počet použitých registrov | Vyžitie GPU |
|---|---|---|
| 1.1 | 32 | 75% |
| 2.0 | 30 | 75% |
| 2.1 | 30 | 75% |
| 2.2 | 32 | 75% |
| 2.3 | 32 | 75% |
| 2.4 | 32 | 75% |
| 3.0 | 36 | 75% |
| 3.6 | 40 | 75% |

vou náročnosťou. Nezáleží celkovo na počte premenných ako skôr na miestach, kde sa používajú. Pri kompilácii sa realizuje taktiež optimalizácia, ktorá redukuje počet využívaných registrov na minimum tým, že sa využívajú miesta v pamäti na dočasné uloženie tých hodnôt, ktoré nie sú aktuálne potrebné. V tabuľke 5.2 sú uvedené počty využitých registrov pri funkcii na GPU, v ktorej sa nachádza hlavná simulácia. Vidíme, že ani pri jednej verzii nám neklesla využiteľnosť GPU pri architektúre Pascal, čo považujeme za dôležitú rolu pri zachovaní výkonnosti. Problém je, že táto verzia je ešte bez driftu, ktorý pridá ďalšiu komplexitu do simulácie a tým pádom taktiež aj prinesie vyššie pamäťové nároky. Už pri využití 41 registrov by klesla využiteľnosť GPU zo 75% na 50%. To by predstavovalo veľmi významnú a citeľnú stratu výkonu GPU. Pri našom aktuálnom využití registrov a zdieľanej pamäte vieme nanajvýš jednu premennú o veľkosti 4 bajtov presunúť do zdieľanej pamäte. To sa ale na samotnom využití registrov vôbec nemusí prejaviť.

Z hľadiska globálnej pamäte sme použili kombináciu štruktúry pre zaznamenia aktuálneho stavu simulácie spolu s globálne definovanou premennou, ktorú sme pomocou atomickej funkcie sčítania inkrementovali. Ten istý princíp sme použili aj pri implementácii [1] jednorozmerných heliosferických modelov. To nám taktiež aj v prípade GPU implementácie 2D F-T modelu umožní dosiahnuť nezávislosť medzi počtom simulácii a veľkosťou polí štruktúr, do ktorých sa ukladajú výsledky. Pri testovaní GPU implementácie 2D F-T modelu sme odmerali veľkosť výsledného súboru s údajmi, ktorý sa pohyboval medzi 25-174.5 GB pre 100 mi-

liárd simulácii v závislosti od vstupných parametrov. Určili sme, že jeden prvok poľa štrúktúr na simuláciu je dostatočný počet vzhľadom k rozsahu vstupných parametrov do simulácie.

Na úrovni zdieľanej pamäti sme vykonali optimalizáciu vzhľadom na architektúru Turing. Pred architektúrou Volta[1] bolo [4] možné alokovať staticky 49152 bajtov na každý blok. Od architektúry Volta je možné alokovať viac ako 49152 bajtov v závislosti od architektúry. Takáto alokácia je možná ale len dynamicky. Harris tvrdí [35], že k tomu, aby sme vedeli využiť inú veľkosť zdieľanej pamäte, ktorá prislúcha jednému bloku, je potrebné použiť funkciu *cudaFuncSetAttribute* s parametrom funkcie, ktorá sa bude vykonávať na GPU, atribútom *cudaFuncAttributeMaxDynamicSharedMemorySize* a veľkosťou zdieľanej pamäte. Táto úprava nám umožnila dosiahnuť využitie grafickej karty architektúry Turing na 100%.

## 5.2 2D F-p a B-p model

K dispozícii pri 2D F-p a 2D B-p modeloch boli verzie modelov, ktorých fyzika je na úrovni verzie 2.4 F-T modelu.

Štruktúra 2D F-p modelu je totožná s 2D F-T modelom. Z hľadiska využitia registrov bolo dosiahnuté pri GPU implementácii 2D F-p modelu také isté využitie, 32 registrov, ako pri GPU implementácii 2D F-T modelu verzie 2.4. Zvýšila sa ale pamäťová náročnosť na globálnu pamäť o približne 10%, keďže je zaznamenávaných viac údajov, ako pri implementácii 2D F-T modelu.

Pri GPU implementácii 2D B-p sme zaznamenali zvýšené použitie registrov. Využitých je až 40 registrov pri aktuálnej verzii implementácie 2D B-p modelu. Využiteľnosť na architektúrach Pascal a Turing to ale nemení. Oproti GPU implementácii 2D F-p a 2D F-T sa zvýšil počet využitých registrov o 7 registrov, ak porovnávame modely, ktoré sú ekvivalentné z úrovne fyziky, ktorá je použitá v týchto modeloch. Pripisujeme to výpočtu energiu w na konci simulácie. Rozdiel medzi modelmi typu forward a backward je taký, že backward vykonáva simulácie spätne v čase, z jedného konkrétneho miesta. Ak by sa fyzika v prípade tohto modelu rozširovala a rozdiel medzi počtom registrov s výpočtom intenzity w a bez výpočtu energie w, je možné presunúť výpočet energie w do ďalšej funkcie.

[1] Architektúra Pascal je [4] priamym predchodcom architektúry Volta, kým architektúra Turing je generačne na rovnakej úrovni ako architektúra Volta.

# 6 Realizácia distribuovaného systému

V tejto kapitole sa budeme venovať implementácii jednotlivých aspektov distribuovaného systému. Pre implementáciu backendu webovej aplikácie a uzlov distribuovaného systému sme sa rozhodli použiť programovací jazyk Java a rámec Spring. Zvolili sme rámec Spring s ohľadom na množstvo integrácii s externými službami, ktoré budeme používať.

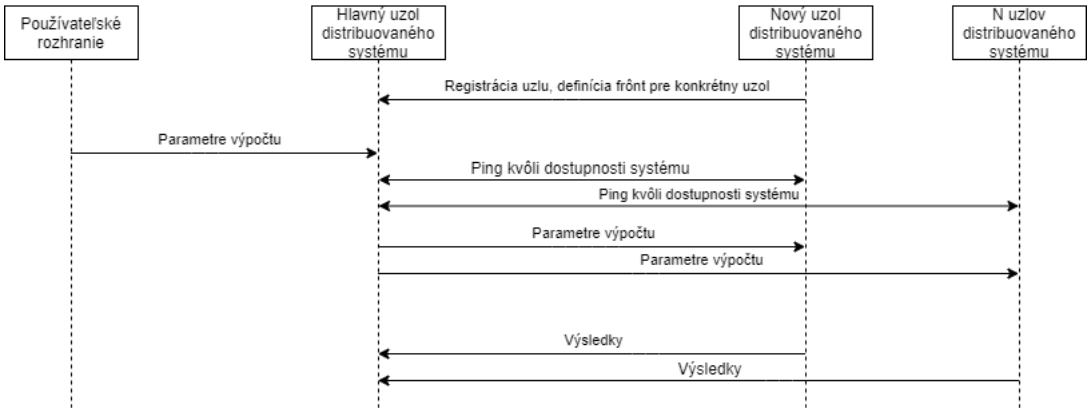
Na úrovni databázy sme zvolili PostgreSQL, kým na úrovni sprostredkovateľa správ(angl. Message broker) RabbitMQ. RabbitMQ sme zvolili [36] z dôvodu perzistentných front, ktoré nám umožnia posilniť toleranciu voči chybám z dôvodu zlyhania backendu webovej aplikácie.

## 6.1 Implementácia hlavného uzlu distribuovaného systému

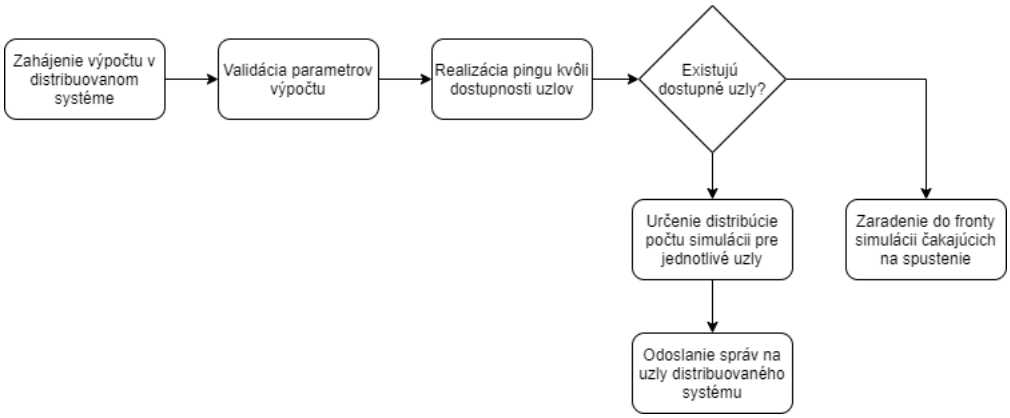
Rozdelili sme backend webovej aplikácie na dve časti a to časť, ktorá predstavuje hlavný uzol distribuovaného systému a časť, ktorá má na starosti komunikáciu s implementáciou používateľského rozhrania.

Na obrázku 6.1 sa nachádza sekvenčný diagram spustenia simulácie na distribuovanom systéme s ohľadom na registráciu nového uzla. Vývojový diagram z hľadiska zahájania simulácie na hlavnom uzle distribuovaného systému sa nachádza na obrázku 6.2. Definovali sme frontu, ktorá slúži na registráciu uzlu distribuovaného systému. Uzol distribuovaného systému posiela názov grafickej karty, ktorú využíva na výpočty, ktorá sa uloží do databázy na neskoršie použitie pri určovaní počtu simulácii, ktoré majú vypočítať jednotlivé uzly distribuovaného systému. V prípade, ak používateľ zadá a odošle parametre výpočtu z používateľského rozhrania, tak backend webovej aplikácie vykoná kontrolu dostupnosti - ping pre všetky aktívne registrované uzly. Následne sú vylúčené tie uzly, ktoré

nie sú dostupné a tie, na ktorých prebiehajú výpočty. Výpočet sa následne rozloží na dostupné grafické karty podľa výpočtovej sily karty, ktorá je dostupná pre daný typ simulácie. Po úspešnom vykonaní simulácii sú následne analyzované výsledky - spektrá odoslané na hlavný uzol, ktorý ich sčíta dohromady.




Obr. 6.1: Sekvenčný diagram spustenia simulácie na distribuovanom systéme




Obr. 6.2: Vývojový diagram zahájenia simulácie na hlavnom uzle

### 6.1.1 Komunikačné prostriedky hlavného uzla distribuovaného systému

Na komunikáciu medzi hlavným uzlom a uzlami distribuovaného systému používame sprostredkovateľ správ(angl. message broker). Definovali sme preto nasledujúce fronty na komunikáciu uzlov distribuovaného systému s hlavným uzlom:

- *client_register* - slúži na registráciu uzla distribuovaného systému, vracia identifikátor UUID, ktorý priradí hlavný uzol uzlu distribuovaného systému,

- *client_update* - slúži na aktualizáciu stavu uzla distribuovaného systému,
- *results* - slúži na zaznamenanie výsledkov, ktoré sú odoslané z jednotlivých uzlov distribuovaného systému.

Z hľadiska REST API sme implementovali endpointy, ktoré slúžia k spúšťaniu simulácii, získaniu prehľadu o stave distribuovaného systému a o detailoch o jednotlivých simuláciach:

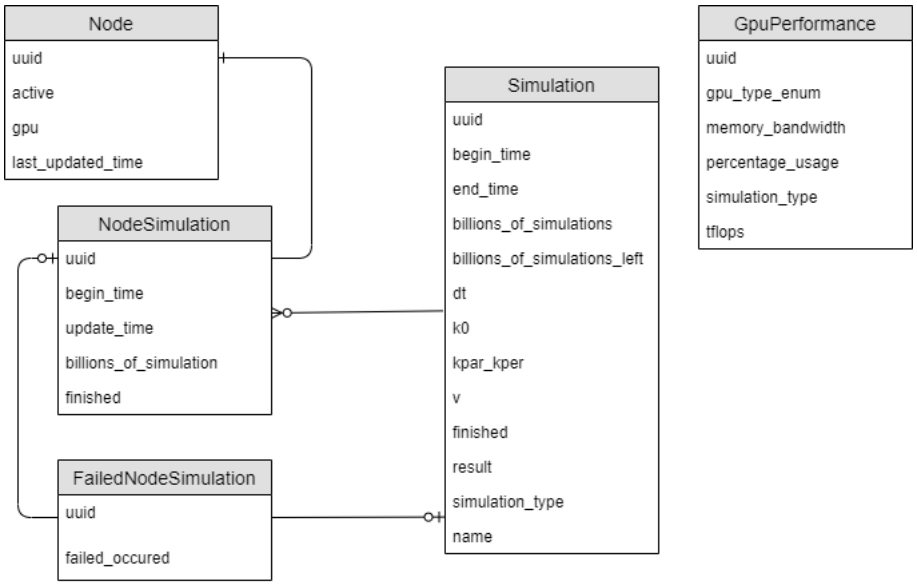
- *POST /simulation/* - slúži na spustenie novej simulácie, resp. zaradenie simulácie do fronty na spracovanie,
- *GET /simulation/all* - vráti zoznam všetkých existujúcich simulácii,
- *GET /simulation/{uuid}* - vráti detailné informácie o simulácii s identifikátorom uuid,
- *GET /simulation/{uuid}/spectrum* - vytvorí a umožní stiahnuť súbor, v ktorom sa nachádza spektrum v .dat alebo .csv formáte.
- *GET /node/all* - vráti zoznam všetkých dostupných uzlov distribuovaného systému a informácie o ich dostupnosti

### 6.1.2 Vzťahy na úrovni databázy

Entitno-relačný diagram na úrovni databázy je uvedený na obrázku 6.3. Definovali sme tabuľku *GpuPerformance* kvôli potrebám výpočtov distribúcie počtu simulácii pre jednotlivé uzly. Obsahuje informácie o type grafickej karty, jej výkonoch a percentuálnom využití pre konkrétny typ simulácie.

Hlavný uzol distribuovaného systému musí mať prehľad o uzloch, ktoré sú zapojené v celom distribuovanom systéme. Preto ukladáme do databázy identifikátor uzla, informácie o tom, či je uzol aktívny, kedy naposledy dorazila správa od uzla distribuovaného systému hlavnému uzlu a aký typ grafickej karty obsahuje.

Samotné simulácie sa nachádzajú v tabuľke *Simulation*. Obsahujú časy, kedy boli dané simulácie spustené a kedy boli ukončené. Obsahujú taktiež parametre simulácii a či sú simulácie ukončené. Tabuľka pre simulácie obsahuje stlpec pre ich výsledky. Pri výbere databázového systému sme uvažovali kvôli výsledkom




Obr. 6.3: Entitno-relačný diagram hlavného uzla distribuovaného systému

uvažovali aj o výbere dokumentovo orientovaného databázového systému. Výsledné spektrá obsahujú približne 1500 záznamov pre tri hodnoty, energia, počet častíc a ich hodnota. Výsledný počet záznamov spektra sa môže odlišovať od typu simulácie. Ak by výsledné spektrá do 150 GeV boli s krokom nie 0.1 GeV, ale 0.01 GeV, spektrum bude obsahovať 15000 hodnôt. Z toho dôvodu sme sa rozhodli použiť dátový typ *jsonb*. Tento typ umožňuje [37] ukladať v údaje vo formáte JSON. To nám umožní reagovať pružne na rôzne typy výsledkov, ktoré len následne vystavime smerom na používateľské rozhranie. Simulácia môže obsahovať viacero simulácii, ktoré sa vykonali, resp. vykonávajú na uzle distribuovaného systému. Nachádzajú sa v tabuľke *NodeSimulation*, pri čom obsahujú identifikátor, začiatočný a posledný čas zmeny, koľko simulácii je pridelených na daný uzol a či simulácie na uzle bola dokončená.

Zaviedli sme tabuľku pre zlyhané simulácie *FailedNodeSimulation*, resp. simulácie, ktoré zlyhali na konkrétnom uzle distribuovaného systému. Definovali sme pri tom čas, kedy nastalo zlyhanie.

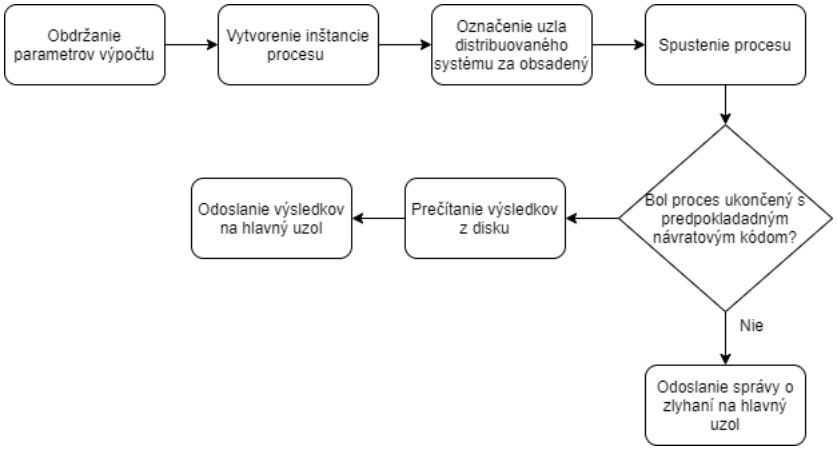
## 6.2 Implementácia uzla distribuovaného systému

Uzol distribuovaného systému by mal plniť rolu prijímania správ od hlavného uzla distribuovaného systému a rolu monitorovania nad procesom, ktorý bude

vykonávať samotné výpočty na GPU.

Každý uzol distribuovaného systému by mal mať pridelený vlastný identifikátor. Na základe neho vytvorí dve fronty:

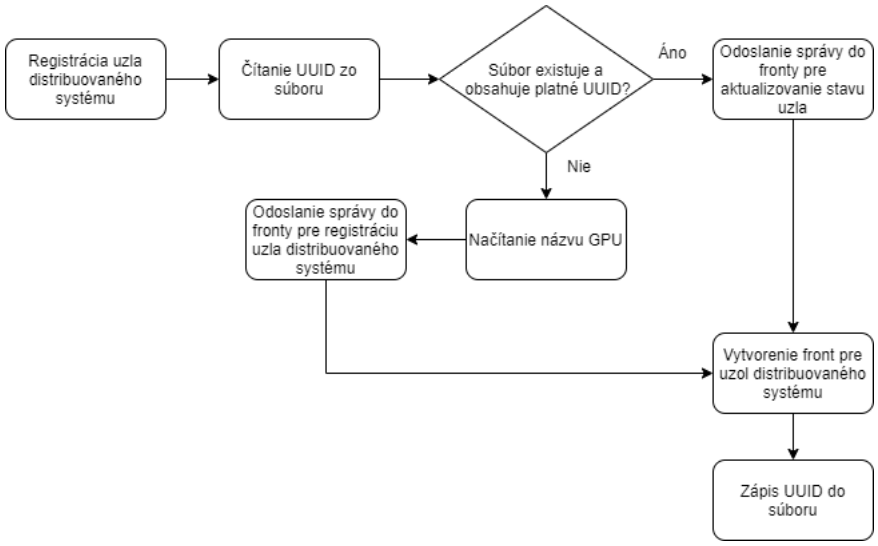
- /{*uuid*}, ktorá slúži na spustenie výpočtov,
- /{*uuid*}/*ping*, ktorá slúži na overenie dostupnosti uzla distribuovaného systému.




Obr. 6.4: Vývojový diagram spustenia výpočtu na uzle distribuovaného systému

Vývojový diagram spustenia výpočtu na uzle distribuovaného systému sa nachádza na obrázku 6.4. Po obdržaní parametrov výpočtov z fronty /{*uuid*} sa na základe obdržaných parametrov vytvorí inštanciu procesu, ktorý má na starosti simulácie distribúcie kozmického žiarenia v heliosfére. Po vytvorení inštancie procesu je uzol distribuovaného systému označený ako obsadený, vyjadrujeme tým stav uzla distribuovaného systému. Po spustení a dokončení požadovaných výpočtov na úrovni procesu kontrolujeme návratový kód procesu. Ak sa nezhodoval s predpokladaným návratovým kódom, tak odosielame na frontu výsledkov správu o zlyhaní výpočtu. Ak sa zhodoval, prečítame výsledné spektrum, ktoré bolo zapísané na disk vrámci vyhodnotenia zaznamenaných výsledkov simulácii. Tie následne odosielame do fronty výsledkov.

Fronta /{*uuid*}/*ping* je určená pre overenie dostupnosti uzla distribuovaného systému. Definovali sme pre tieto účely komponent, v ktorom udržiavame aktuálny stav distribuovaného systému z hľadiska dostupnosti uzla. Dostupnosť reprezentujeme binárnou hodnotou.




Obr. 6.5: Vývojový diagram registrácie uzla distribuovaného systému

Na obrázku 6.5 sa nachádza vývojový diagram registrácie uzla distribuovaného systému. Uzol distribuovaného systému by mal byť registrovaný v hlavnom uzle iba raz. Preto identifikátor UUID zapisujeme a čítame zo súboru. Ak súbor existuje a obsahuje validné UUID, je odoslaná správa do fronty pre aktualizovanie stavu uzla distribuovaného systému. Ak nie, tak načíta názov GPU, ktorú bude používať uzol. Pre správnu funkcionalitu distribúcie počtu simulácii jednotlivým uzlom distribuovaného systému je potrebné, aby hlavný uzol distribuovaného systému vedel, aké GPU sú k dispozícii na jednotlivých uzloch. Pre detekciu GPU sme sa rozhodli použiť nástroj *nvidia-smi*. Umožňuje zobraziť [38] zoznam dostupných GPU v systéme. Z konzolového výstupu procesu sme načítali a pomocou regulárnych výrazov zistili názov GPU, ktorú bude uzol distribuovaného systému používať. Následne posielame názov GPU do fronty určenej na registráciu. Pomocou obdržaného indetifikátora sú definované fronty pre spustenie výpočtu a pre overenie dostupnosti uzla. Identifikátor zapíšeme na disk pre neskoršie použitie. Registrácia sa opakuje periodicky, kým nie je úspešná.

## 6.3 Distribúcia počtu simulácii uzlom distribuovaného systému

Jedným z problémov, ktorý sa vyskytol bolo to, ako budeme deliť počty simulácii jednotlivým uzlom. Analyzovali sme preto možnosti, aby sme dosiahli optimálnu distribúciu výpočtov v distribuovanom systéme.

Funkcionalitu, ktorú chceme docieliť poskytuje [39] OpenMP vo forme dynamického plánovanie(angl. dynamic scheduling). To nám umožňuje rozdeliť akýkoľvek cyklus for dynamicky podľa potreby pre jednotlivé vlákna tak, aby boli čo najviac rovnomerne vyťažené. Analogicky, ak by sme chceli rozdeliť 100 miliárd simulácii z akéhokoľvek modelu typu forward, tak by sme mohli deliť simulácie rovnomerne po 10 iteráciach pre každý uzol distribuovaného systému. Referenčné systémy, na ktorých bude nasadený uzol distribuovaného systému majú rozlične výkonné grafické karty. Mohlo by dôjst k situácii, kedy by bolo posledný 10 miliárd simulácii pridelených grafickej karte, no druhá karta by krátko po takomto pridelení skončila svoju činnosť a ostala bez pridelenej úlohy. Tento problém by bol riešiteľný tým, s menším počtom iterácii by sme prideľovali menší počet na výpočty - čo by sme mohli reprezentovať napríklad binárnym stromom. Tento prístup je problematický v tom, že používame používateľské rozhranie v príkazovom riadku na interakciu s implementáciou určenou pre GPU. Inicializácia generátorov náhodných čísel trvá približne 4.5 minúty na GTX 1080 TI a 9 minút na RTX 2060. To predstavuje pri prideľovaní 10 milárd simulácii približne 32-36 minútovú stratu, ktorú predstavuje inicializácia generátorov náhodných čísel. Výpočet na GTX 1080 TI pre 100 miliárd simulácii 1D F-p modelu trval [1] 5 hodín a 37 minút. Mohli by sme implementovať medziprocesovú komunikáciu, cez ktorú by sme posielali počty simulácii, ktoré chceme vykonať, ale na úrovni implementácii modelu by to nemuselo priniesť želaný výsledok a došlo by ku skomplikovaniu abstrakcie, nad ktorou sa pri modeli pracuje.

Rozhodli sme sa preto aplikovať iný prístup. Vieme, aké grafické karty na výpočty používajú jednotlivé uzly distribuovaného systému. Implementácie modelov sú silno optimalizované na konkrétne architektúry GPU od spoločnosti Nvidia. Dostatočná štatistika pre forward modely je približne 100 miliárd simulácii na to, aby sme vedeli vylúčiť nedostatočnú štatistiku ako dôvod odchýliek na jednotlivých energiách energetického spektra výsledkov simulácii. To nás vedie

Tabuľka 6.1: Vykonávacie časy 1D F-p modelu pre 100 miliárd simulácii s dĺžkou kroku dt = 5.0 s pôvodnou distribúciou simulácii

| Grafická karta | Počet simulácii[mld] | Vykonávací čas[min] |
|---|---|---|
| GTX 1080 TI | 57 | 238 |
| RTX 2060 | 43 | 328 |

k možnosti rozdeľovať počty simulácii na základe vyjadreného výpočtového výkonu GPU, ktorý by sme prenásobili percentuálnym využitím GPU pri funkcii, ktorá vykonáva samotnú simuláciu.

Vzorec, ktorý bol použitý pri výpočte teoretického výkonu je uvedený v rovnici 6.1. Do výpočtu sme vzali do úvahy maximálny výkon karty pri 32-bitových číslach s pohyblivou rádovou čiarkou. Predpokladali sme, že vzhľadom k tomu, že prevažnú časť vykonávacieho času tvorí vykonávací čas samotných simulácii, tak to bude postačujúce k vyjadreniu výkonnosti karty vzhľadom k simuláciam kozmického žiarenia v heliosfére. Pre test sme si zvolili jednorozmerný F-p model pre 100 miliárd simulácii. V tabuľke 6.1 sa nachádzajú vykonávacie časy pre 1D F-p model pre 100 miliárd simulácii a dĺžku kroku dt = 5.0. Medzi vykonávacím časom na GTX 1080 TI a RTX 2060 je 31.80% rozdiel. Zároveň ale celkový čas vykonávania v distribuovanom systéme je 328 minút na dvoch GPU, čo predstavuje minimálny nárast zrýchlenia oproti vykonávaciemu času s rovnakými vstupmi pre 1D F-p model na referenčnom počítači s GTX 1080 TI, ktorý predstavuje [1] 337 minút.

$$Vykon = VykonTFLOPS * VyuzitieGPU \tag{6.1}$$

Začali sme hľadať chybu, prečo je náš vzorec nesprávny, resp. nie úplne kompletný. Pri porovnávaní výkonnosti sme zistili, že RTX 2060 je oproti GTX 1080 TI o približne 50% pomalšia. Zároveň sme si všimli to, že na referenčnom počítači s RTX 2060 bol vykonávací čas pre proces, ktorý manažuje zjednotenú pamäť, vyšší ako vykonávací čas o niekoľko sekúnd, ako sme pozorovali pri referenčnom počítači s GTX 1080. Zjednotená pamäť predstavuje len zlomok údajov, ktoré sa presúvajú medzi pamäťou RAM a globálnou pamäťou grafickej karty. Do vzorca, ktorý vypočíta výkonnosť, resp. výkonnostnú hodnotu danej grafickej karty, sme sa rozhodli doplniť priepustnosť pamäte grafickej karty. Aktualizovaný vzorec je

Tabuľka 6.2: Vykonávacie časy 1D F-p modelu pre 100 miliárd simulácii s dĺžkou kroku dt = 5.0 s vylepšenou distribúciou simulácii

| Grafická karta | Počet simulácii[mld] | Vykonávací čas[min] |
|---|---|---|
| GTX 1080 TI | 64 | 251 |
| RTX 2060 | 36 | 257 |

uvedený v rovnici 6.2. Opakovali sme testovanie pre GPU implementáciu 1D F-p modelu pre 100 miliárd simulácii a dĺžku kroku dt = 5.0. V tabuľke 6.2 sa nachádzajú vykonávacie časy pre 1D F-p model pre 100 miliárd simulácii a dĺžku kroku dt = 5.0 po vykonaní korekcie vo vzorci. Rozdiel vo vykonávacích časoch predstavuje 2.36%. Oproti vykonávaniu všetkých simulácii na GTX 1080 TI to predstavuje 26.93% nárast výkonnosti.

$$Vykon = VykonTFLOPS * VyuzitieGPU * PriepustnostGPUPamate \quad (6.2)$$

Distribúcia bola realizovaná najmä s ohľadom na modely typu forward, pri ktorých výpočet trvá na GPU výrazne dlhšie, ako v prípade modelov typu backward.

## 6.4 Tolerancia chýb na úrovni distribuovaného systému

Pri tolerancii chýb a zotaveniu sme zamerali našu pozornosť najmä na to, aby výpočty simulácii boli dokončené už bez ohľadu na rýchlosť z hľadiska delenia. Preto sme implementovali nasledujúce prípady do mechanizmov tolerancie chýb distribuovaného systému:

- prípad chýb, ktoré by mohli nastať počas vykonávania simulácii kozmického žiarenia v heliosfére,
- prípad, ak zlyhá sieť medzi MOM(angl. Message oriented middleware) a uzlom distribuovaného systému, resp. ak zlyhá samotný uzol distribuovaného systému,

V prípade chýb, ktoré by mohli nastať počas vykonávania simulácii kozmického žiarenia v heliosfére sme implementovali mechanizmus, ktorý v prípade akéhokoľvek zlyhania na úrovni procesu odchytí a odošle hlavnému uzlu distribuovaného systému správu o tom, že vypočet požadovanej simulácie zlyhal a zmení svoj stav aktívnosti výpočtu na neaktívny. Na hlavnom uzle je následne časť simulácie zapísaná medzi zlyhané simulácie.

Ako sme si uviedli v kapitole 3.5, distribuovaný systém by mal vedieť rozoznať, či sa jedná o zlyhanie na úrovni siete alebo na úrovni softvéru. Rozoznať úplne tieto problémy je pomerne ťažké v našom prípade. Detekciu zlyhania na úrovni softvéru z hľadiska zlyhania výpočtu na GPU vieme rozoznať. Na úrovni zlyhania uzla distribuovaného systému by sme potrebovali funkcionalitu na úrovni operačného systému. Operačný systém bude spustený v Docker kontajneri. To nás vedie k tomu, že nie celkom vieme rozoznať, či sa jedná o softvérovú chybu, alebo chybu siete. Taktiež v prípade chyby v sieti sa môže jednať o dlhodobý výpadok.

Preto sme sa rozhodli implementovať detekčný vzor heartbeat. Spočíva v periodickom odosielaní [40] odosielaní správ z strany klienta alebo servera. Klient, resp. server, ktorý prijíma tieto správy má definovaný čas, po ktorého uplynutí je spojenie vyhlásené za prerušené. Rozhodli sme sa pre implementáciu na strane uzlu distribuovaného systému, ktorý periodicky posiela správy na hlavný uzol distribuovaného systému s periodicitou 2 minút. Pre tento účel sme definovali samostatnú frontu. Hlavný uzol distribuovaného systému si pri prijímaní správy aktualizuje čas poslednej aktualizácie. Na základe neho každých 15 minút kontroluje, ktoré uzly majú čas poslednej aktualizácie starší ako 15 minút. Ak ich čas je starší ako 15 minút, tak sú označené ako neaktívne. Pokiaľ pošle uzol distribuovaného systému, ktorý bol pred tým označený ako neaktívny správu, je znova označený ako aktívny.

## 6.5 Webové používateľské rozhranie

Webové používateľské rozhranie sme implementovali s dôrazom na ušetrenie času prístupu k výsledkom. Pri implementácii sme štruktúru webového používateľského rozhranie rozdelili z hľadiska funkcionality na:

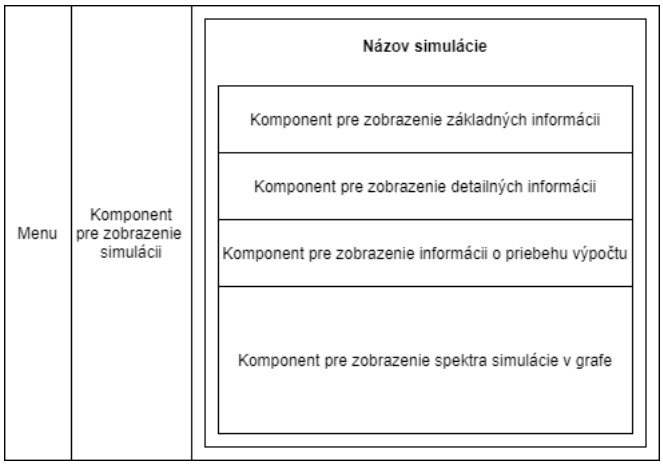
- komponent pre spustenie výpočtov pre jednotlivé typy simulácii,

- komponent pre zobrazenie informácii o uzloch,
- komponent pre výsledky simulácii,
  - komponent pre zobrazenie základných informácii o simulácii,
  - komponent pre zobrazenie detailných informácii o simulácii,
  - komponent pre zobrazenie informácii o priebehu výpočtu,
  - komponent pre zobrazenie spektra simulácie v grafe.

Principiálne rozloženie komponentov, ktoré sa nachádzajú na obrazovke, ktorá slúži na spustenie výpočtov, je znázornené na obrázku 6.6. Používateľovi sa okrem možnosti spustenia výpočtu zobrazia aj informácie o jednotlivých uzloch distribuovaného systému a ich stav.


Obr. 6.6: Principiálne rozloženie komponentov obrazovky pre spustenie výpočtu

Principiálne rozloženie komponentov, ktoré sa nachádzajú na obrazovke pre zobrazenie výsledkov simulácii sa nachádza na obrázku 6.7. Komponent pre zobrazenie simulácii zobrazuje zoznam všetkých dostupných simulácii. Po kliknutí na konkrétnu simuláciu sa zobrazia detaily danej simulácie. V detaile sme implementovali komponent pre zobrazenie základných informácii o simulácii - názov, kedy bola spustená, ukončená. V komponente pre zobrazenie detailu o simulácii zobrazujeme v tabuľke fyzikálne parametre danej simulácie. Považovali sme taktiež nutné informovať používateľa o priebehu výpočtu, ako sa delí záťaž medzi jednotlivé uzly distribuovaného systému. Môže sa zdať, že sa jedná o porušenie




Obr. 6.7: Principiálne rozloženie komponentov obrazovky pre zobrazenie výsledkov simulácii

základných charakteristík distribuovaných systémov, ktoré sme si uviedli v kapitole 3.3. Používateľ je len informovaný o rozložení výpočtu na jednotlivé uzly a o vykonávacom čase na jednotlivých uzloch, nemá to žiadny iný vplyv pri zadávaní výpočtu.

# 7 Dosiahnuté výsledky

Našim cieľom v tejto kapitole je analyzovať výsledné spektrá z modifikovaného 1D GPU F-p modelu so zmenami popísanými kapitole 4.1. Ďalej sa budeme venovať dosiahnutému zrýchleniu, presnosti a analýze spektier z 2D F-T modelu verzie 2.3 a 3.6.

## 7.1 Analýza spektier z 1D GPU F-p modelu

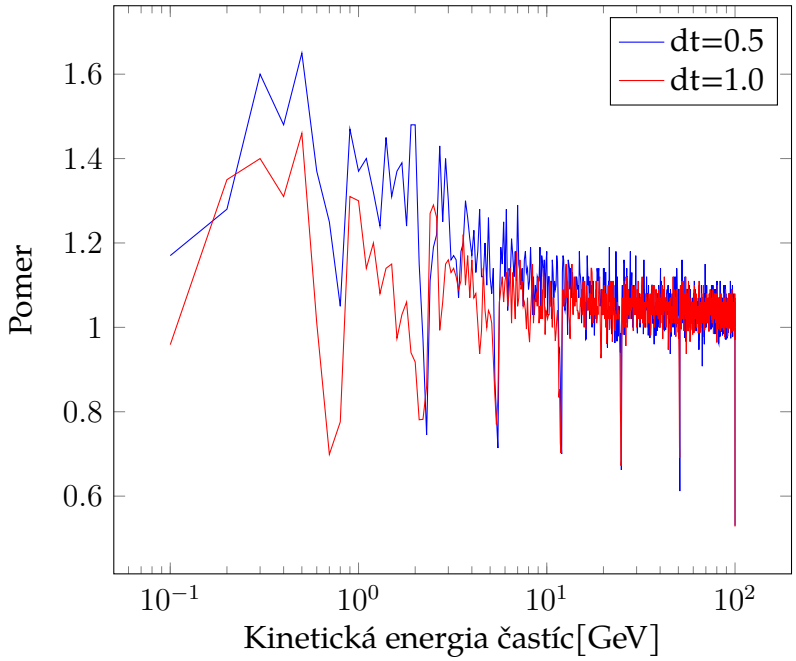
V kapitole 4.1 sme navrhli spôsob, ako budeme testovať jednotlivé oblasti, ktoré mohli spôsobovať pulzy v 1D F-p modeli.

Pri testovaní zmeny generátora náhodných čísel XORWOW za generátor náhodných čísel Philox sme pre simulácie implementácie 1D F-p modelu so vstupnou dĺžkou kroku dt = 0.5 nezaznamenali zmeny v spektre, t.j. pulzy ostali na energiách na ktorých boli s rovnakou veľkosťou. Testovanie zmeny distribúcie počtu simulácii pre implementáciu 1D F-p modelu so vstupnou dĺžkou kroku dt = 0.5 dopadlo podobne a výsledný pomer spektra je totožný so spektrom uvedeným v grafe 2.1.

Pred tým, ako sme pristúpili k odstráneniu prepínača *–use_fast_math* sme sa rozhodli pristúpiť k ďalšiemu testovaniu pulzov pri implementácii 1D F-p modelu na GPU a to z toho dôvodu, že odstránenie optimalizačného prepínača *–use_fast_math* spôsobí narást vykonávacieho času. Preto sme rozhodli nájsť časový krok dt taký, aby bol potencionálny vykonávací čas pre nás prijateľný a pulzy boli dostatočne viditeľné a výrazné.

V grafe 7.1 sa nachádza pomer energetických spektier 1D B-p a 1D F-p modelu s dĺžkou kroku dt = 0.5 a dt = 1.0. Vidíme, že pulzy sa nachádzajú na veľmi podobných miestach. Pulz na 0.4-0.6 GeV je oveľa viac viditeľnejší pre dĺžku časového kroku dt = 1.0. Pulz medzi 10 - 20 GeV je takmer totožný - nábežná hrana

Obr. 7.1: Pomer energetických spektier 1D B-p a 1D F-p modelu s dt = 0.5 a dt = 1.0 pre 200 miliárd častíc



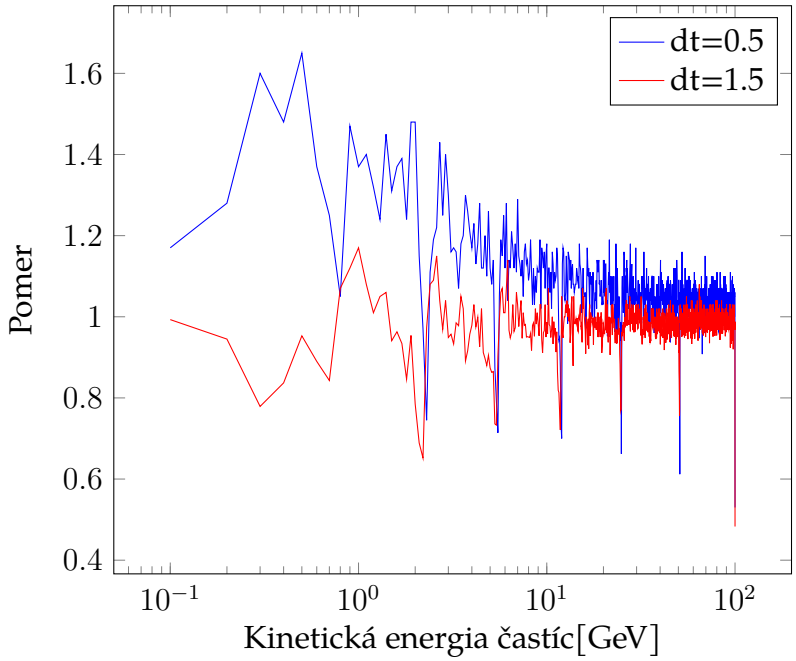

pre dĺžku kroku dt = 1.0 ma iný sklon ako nábežná hrana pre dĺžku kroku dt = 0.5.

Keďže testovanie s dĺžkou kroku dt = 1.0 vykazovalo ďalej chybu, rozhodli sme sa vykonať testovanie pre dĺžku kroku dt = 1.5. V grafe 7.2 sa nachádza pomer energetických spektier 1D B-p a 1D F-p modelu s dĺžkou kroku dt = 0.5 a dt = 1.5. Pulz medzi 0.4-0.6 GeV nie je viditeľný ako pri pomere energetického spektra 1D F-p modelu s dĺžkou kroku dt = 1.0. Pulz medzi 2-3 GeV je o mnoho výraznejší ako pre prípad s dĺžkou kroku dt = 0.5. Pulzy sú dostatočne viditeľné.

Rozhodli sme sa preto zvýšiť dĺžku časového kroku dt o 0.5 na dt = 2.0. V grafe 7.3 sa nachádza pomer energetických spektier 1D B-p a 1D F-p modelu s dĺžkou kroku dt = 0.5 a dt = 2.0. Pulzy už nie sú viditeľné. Pre overenie, že pulzy sa nevyskytujú pre dĺžky časových krokov dt > 2.0, sme vykonali testovanie aj pre simuláciu 200 miliárd častíc v 1D F-p modeli s dĺžkou časového kroku dt = 2.5. V grafe 7.4 vidíme, že pulzácia tak ako pri dĺžke časového kroku dt = 2.0 nie je viditeľná.

Odstránenie optimalizačného prepínača *–use_fast_math* spôsobí to, že vykonávací čas sa značne predlží. Preto pre určenie vhodnej dĺžky časového kroku dt

Obr. 7.2: Pomer energetických spektier 1D B-p a 1D F-p modelu s dt = 0.5 a dt = 1.5 pre 200 miliárd častíc

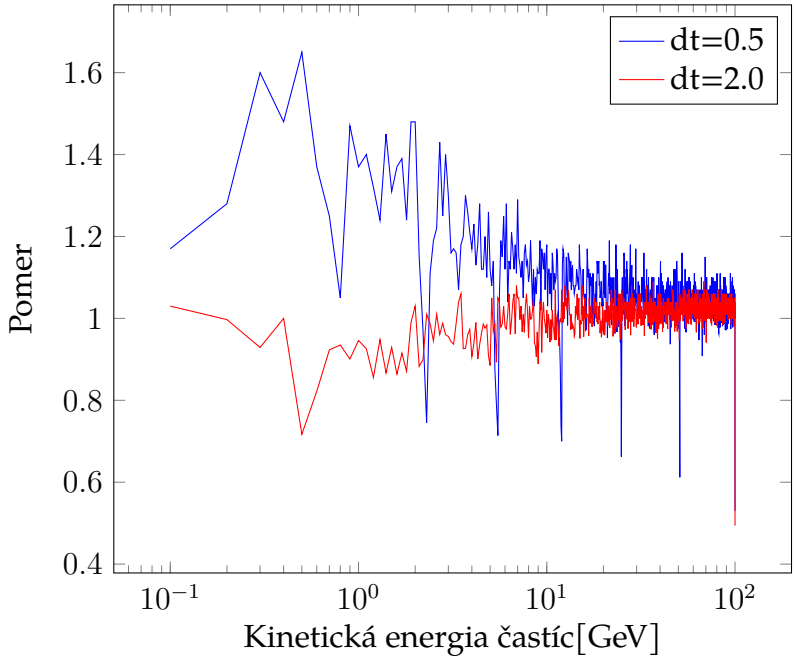



sme vykonali ešte testovanie pre dĺžku časového kroku dt = 1.75. V grafe 7.5 sa nachádza pomer energetických spektier 1D B-p a 1D F-p modelu s dĺžkou kroku dt = 0.5 a dt = 1.75. Pulzácia je prítomná, ale v energiách vyšších ako 10 GeV sú pulzy jednoznačne menšie, až o polovicu veľkosti pulzu na danej energii oproti pulzu z energetického spektra s dĺžkou časového kroku dt = 0.5. Rozdiel medzi pulzom na úrovni 2-3 GeV je 0.04.

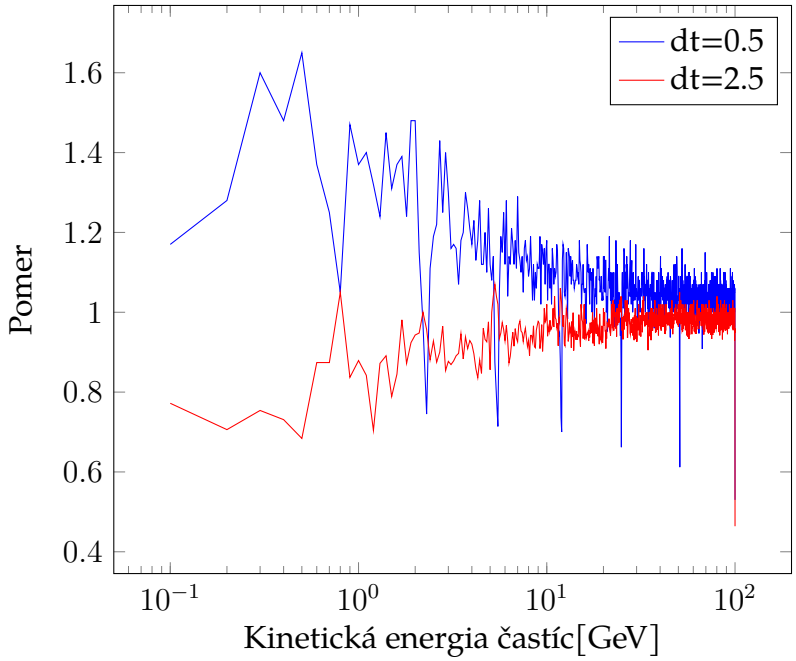
Z testov nám vyplýva, že kým pri dĺžke časového kroku dt = 2.0 sa nenachádzajú pulzy, tak pri dĺžke časového kroku dt = 1.75 sú viditeľné pulzy, ale vo veľkosti jednotlivých pulzov sú viditeľne o mnoho menšie ako pulzy vyskytujúce sa pri dĺžke časového kroku dt = 0.5. Preto sme sa rozhodli realizovať testovanie s odstránením prepínača *–use_fast_math* použiť dĺžku kroku dt = 1.5. Výpočet s použitím optimalizačného prepínača *–use_fast_math* trval 19.25 hodiny pre 200 miliárd simulácii. Výpočet s odstránením optimalizačného prepínača *–use_fast_math* trval viac ako 128 hodín pri rovnakej štatistike. Tento 6.65-násobný rozdiel bol spôsobený využitím softvérovej implementácie matematických funkcií miesto využitia hardvérových prostriedkov, ktoré dokázali riešiť matematické funkcie.

V grafe 7.6 sa nachádza pomer energetických spektier 1D B-p a 1D F-p mo-

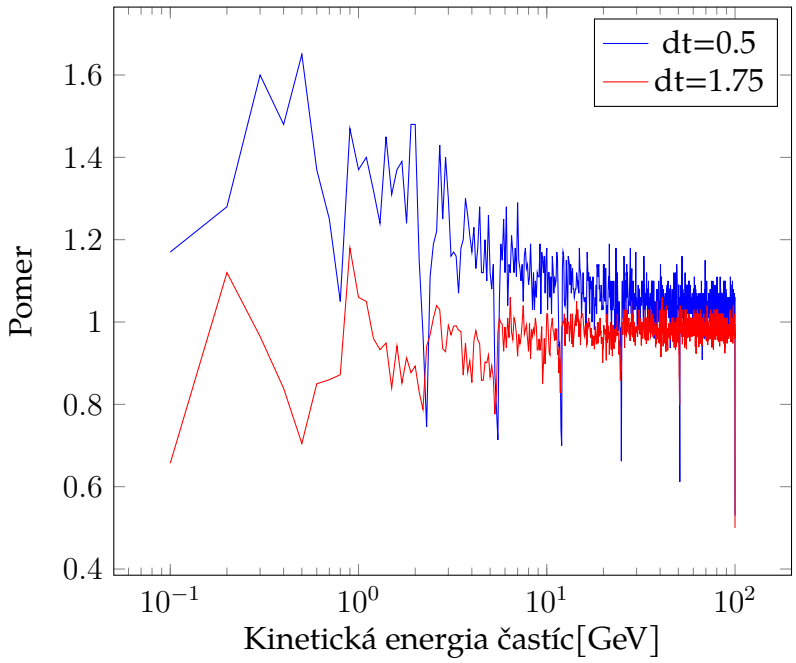
Obr. 7.3: Pomer energetických spektier 1D B-p a 1D F-p modelu s dt = 0.5 a dt = 2.0 pre 200 miliárd častíc



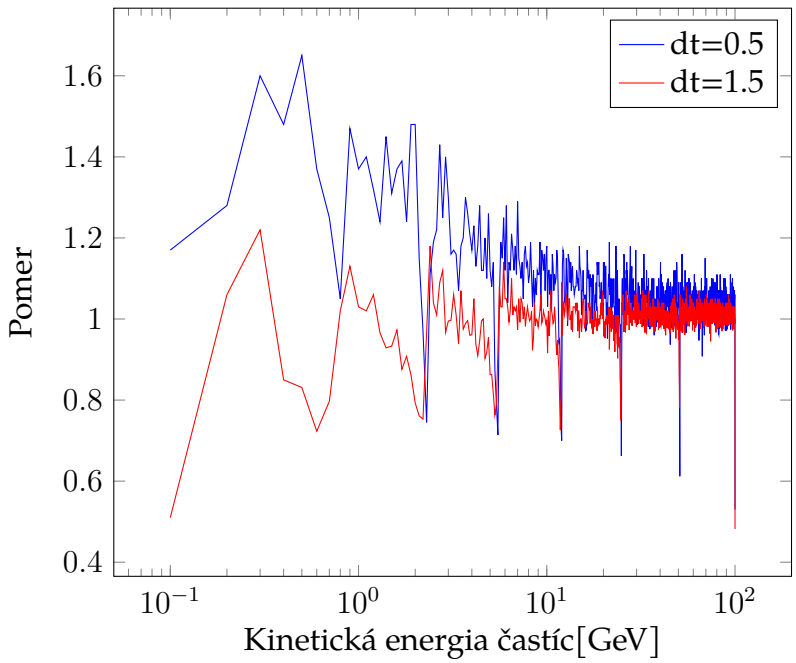

Obr. 7.4: Pomer energetických spektier 1D B-p a 1D F-p modelu s dt = 0.5 a dt = 2.5 pre 200 miliárd častíc



Obr. 7.5: Pomer energetických spektier 1D B-p a 1D F-p modelu s dt = 0.5 a dt = 1.75 pre 200 miliárd častíc




Obr. 7.6: Pomer energetických spektier 1D B-p a 1D F-p modelu s dt = 0.5 a dt = 1.5 bez použitia optimalizačného prepínača *–use_fast_math* pre 200 miliárd častíc



delu s dĺžkou kroku dt = 0.5 a dt = 1.5 bez použitia optimalizačného prepínača *–use_fast_math*. Pulzácia je o mnoho jasnejšie viditeľná aj do 1 GeV, čo s použitím optimalizačného prepínača *–use_fast_math* nebola jednoznačne viditeľná. Vyplýva z toho to, že tieto pulzy sú pravdepodobne spôsobené rozdielmi medzi dvojitou a jednoduchou presnosťou. Toto tvrdenie ale nevieme z časového hľadiska testovať, keďže výpočet s implementácie 1D F-p modelu s dvojitou presnsťou na GPU pre dĺžku časového kroku dt = 0.5 a pre štatistiku 200 milárd simulácii by trval viac ako 6 mesiacov.

## 7.2 Modely typu Forward

V nasledujúcich podkapitolách sa budeme venovať zrýchleniu, presnosti a analýze spektier získaných z GPU implemenetácie 2D F-T modelu vo verzii 2.3 a 2D B-p modelu. Fyzika, ktorú obsahuje 2D F-T model verzie 2.3 je výrazne jednoduchšia oproti verzii 3.6, preto sme sa rozhodli ju využiť na testovanie základných vlastností modelu.

V prípade 2D B-p modelu sme mali k dispozícii model, ktorého úroveň fyziky zodpovedá úrovni fyzike 2D F-T modelu verzie 2.3.

Spektrá z GPU implementácia forward modelov bude potrebné testovať, aby bolo možné určiť limity a ich presnosť. Vzhľadom k množstvu parametrov a ich krajným hodnotám boli vybrané hodnoty vstupných parametrov, ktoré sú uvedené v tabuľke 7.1. Pri testoch I. a II. budú testované krajné hodnoty parametra pomeru $K_{per}$ a $K_{par}$, podobne v prípade testov III. a IV. budú testované krajné hodnoty parametra rýchlosti V. V prípade testov V. a VI. bude sledované správanie pomerov výsledného spektra z GPU implementácia a referenčného spektra v prípade maximálnej modulácie.

### 7.2.1 Zrýchlenie získané pri GPU implementácii 2D F-T modelu verzie 2.3

Meranie zrýchlenia sme realizovali pri vstupných parametroch dĺžky kroku dt = 5.0s, $K_0 = 5 * 10^{18}$ $m^2/s$, rýchlosti V = 400km/h a pomere $K_{per}$ a $K_{par} = 0.01$ pri štatistike 100 miliárd simulácii. Tieto hodnoty nevyjadrujú krajné možné hodnoty, ktoré budú vstupom do tohto modelu. Oproti GPU implementácii 1D F-p

Tabuľka 7.1: Vstupné parametre pre testovanie GPU implementácii typu forward

| Označenie testu | dt | V | K0 | Pomer $K_{per}$ a $K_{par}$ |
|---|---|---|---|---|
| I. | 5.0 | 400 | $5 * 10^{18}$ | 0.01 |
| II. | 5.0 | 400 | $5 * 10^{18}$ | 0.1 |
| III. | 5.0 | 300 | $5 * 10^{18}$ | 0.01 |
| IV. | 5.0 | 700 | $5 * 10^{18}$ | 0.01 |
| V. | 5.0 | 700 | $1 * 10^{18}$ | 0.01 |
| VI. | 0.5 | 700 | $1 * 10^{18}$ | 0.01 |

modelu GPU implementácia 2D F-T modelu produkuje [1] až 213-násobne viac údajov - výsledný súbor s údajmi pre implementáciu 1D F-p modelu mal 600MB a výsledný súbor s údajmi pre implementáciu 2D F-T modelu mal 128GB. Vykonávací čas pre GPU implementáciu 2D F-T modelu bol 15 hodín a 10 minút. Ak vezmeme do úvahy, že priemerne sme dokázali na pevný disk zapisovať s rýchlosťou 70MB/s, tak len zápis na disk trval 30 minút a 28 sekúnd - t.j 3.3% celkového vykonávacieho času.

Na referenčnom multiprocesorovom systéme trval výpočet pre 100 miliárd simulácii implementácie 2D F-T modelu 150 hodín na 64 jadrách CPU. Výpočet sa spúšťal cez príslušný počet procesov - každý na jedno jadro CPU. Oproti tomuto systému sme získali 9.83-násobné zrýchlenie v najhorším prípade pomocou implemenetácie modelu na GPU. Toto zrýchlenie je veľmi podobné tomu, ktoré sme získali [1] pri GPU implementácii 1D F-p modelu - 7.87-násobné zrýchlenie oproti rovnakému multiprocesorovému systému.

Na inom multiprocesorovom systéme trval výpočet 100 miliárd simulácii implemenetácie 2D F-T modelu 240 hodín pri použití 20 jadier CPU. Oproti tomuto systému sme získali 15.82-násobné zrýchlenie na GPU implemenetácii 2D F-T modelu.

### 7.2.2 Vyhodnotenie presnosti pri GPU implementácii 2D F-T modelu verzie 2.3

V prípade všetkých testov, ktoré boli určené v kapitole 7.2, sme sledovali, aké množstvo údajov zapíše GPU implementácia 2D F-T modelu verzie 2.3 na pevný

Tabuľka 7.2: Výstupné množstvo údajov z jednotlivých testov pri štatistike 100 miliárd simulácii

| Označenie testu | Množstvo údajov[GB] |
|---|---|
| I. | 127.4 |
| II. | 25.0 |
| III. | 95.2 |
| IV. | 174.5 |
| V. | 165.7 |
| VI. | 171.3 |

disk. Množstvá jednotlivých výsledkov sme uviedli v tabuľke 7.2. Usudzujeme z toho, že najväčší vplyv na množstvo údajov má pomer $K_{per}$ a $K_{par}$, keďže rozdiel medzi maximálnou a minimálnou bol až 102.4 GB, pri čom bol výsledný súbor 5.096-krát menší. Ďalší veľmi výrazný vplyv predstavoval vstupný parameter rýchlosti V, pri čom rozdiel medzi minimálnou a maximálnou hodnotou predstavoval 79.3 GB. Oproti maximálnej hodnote rýchlosti V bol výstupný súbor s minimálnou hodnotou rýchlosti V 1.832-krát menší. Ani tieto rozdiely pri štatistike 100 miliárd simulácii nám nerobili problémy - výsledná štatistika bola dostatočná na posúdenie vplyvu presnosti.

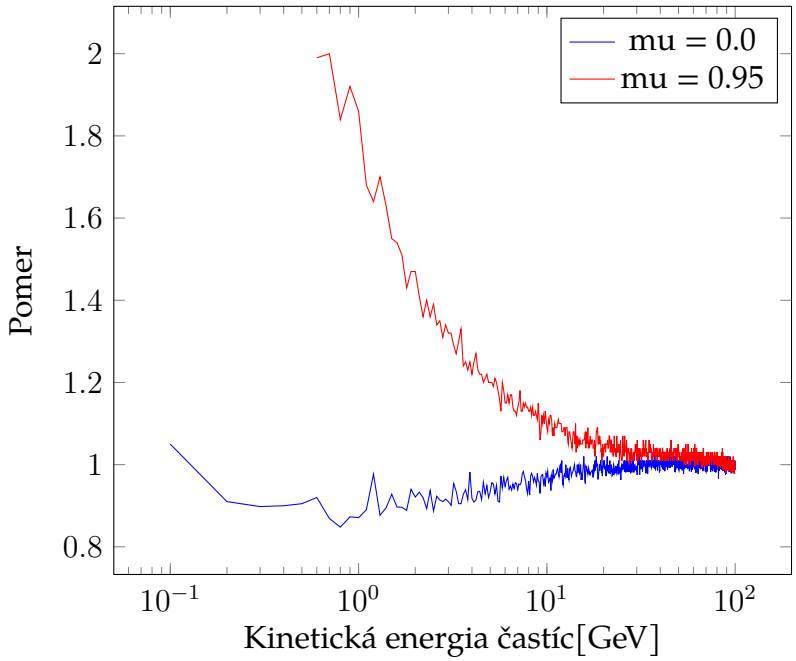
Na obrázku 7.7 sa nachádza pomer energetických spektier GPU implementácie 2D F-T a 2D B-p modelu pre test I. Kým pre mu = 0.0 je rozdiel najviac na úrovni 16% na 0.8 GeV, pre mu = 0.95 klesol rozdiel pod 1.5 až pri 1.8 GeV. Zníženie rozdielu pod úroveň 15% pre mu = 0.95 nastáva pri 6.7 GeV.

Pre test II, ktorého pomer výsledných energetických spektier 2D F-T a 2D B-p modelu sa nachádza na obrázku 7.8, boli zaznamenané len veľmi mierne odchýlky na úrovni maximálne 9% pre mu = 0.95. Pre GPU implementáciu 2D F-T modelu je to jediný test, pri ktorom pomer spektier pre mu = 0.95 bol dostatočne presný, bez výskytov rozdielov nad 2 na nízkych energiách. Pre mu = 0.0 je najvyšší rozdiel na úrovni 26% pri 0.0 GeV, na energiách vyšších ako 2.6 GeV sa dostáva pod 10%.

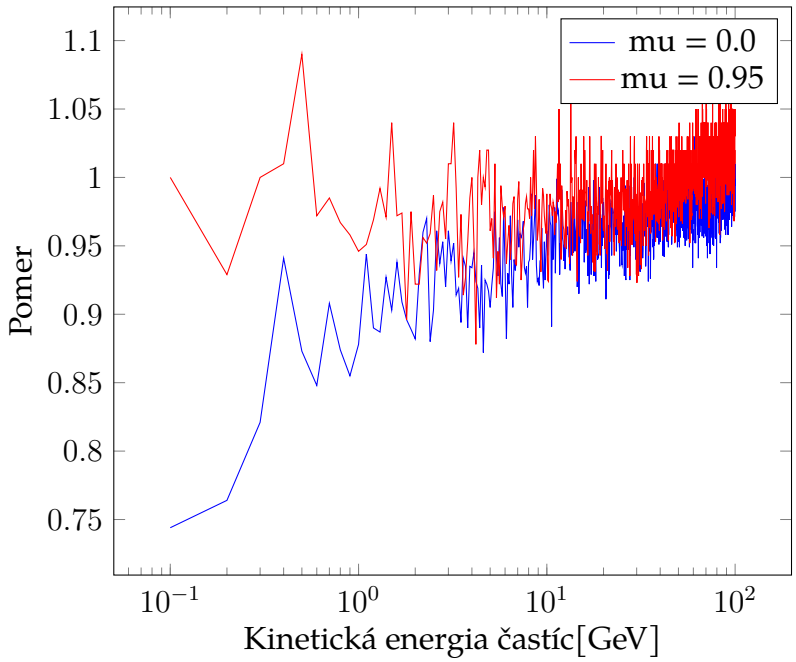
Na obrázku 7.9 sa nachádza pomer energetických spektier GPU implementácie 2D F-T a 2D B-p modelu pre test III. Pre mu = 0.0 sa najvyšší rozdiel nachádza na 1 GeV, a to 19.1%. Pre mu = 0.95 klesol rozdiel pod 1.5 pri 1.5 GeV.

Pri teste IV., ktorého pomer energetických spektier GPU implementácie 2D F-

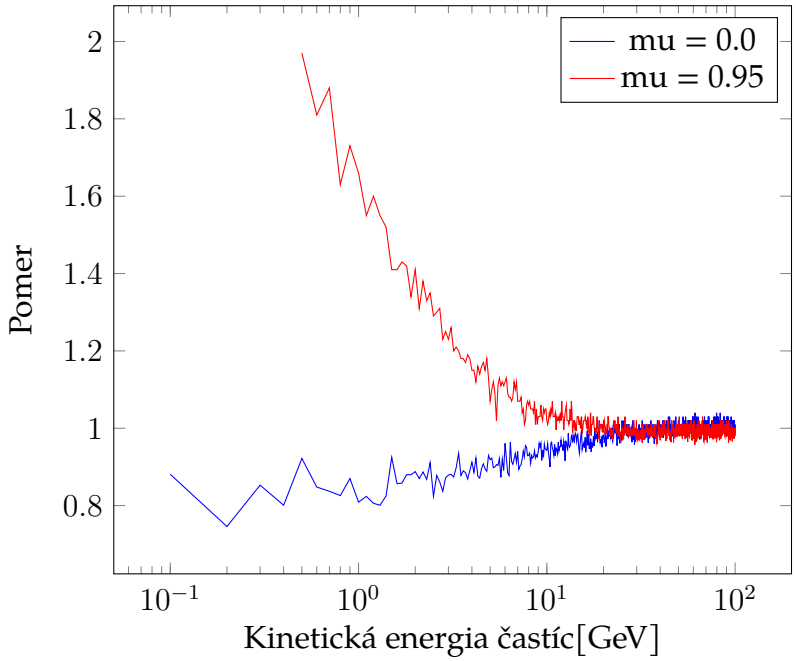
Obr. 7.7: Pomer energetických spektier 2D F-T a 2D B-p modelu s dt = 5.0 s, $K_0$ = 5 * $10^{18}$ $m^2/s$, rýchlosti V = 400km/h a pomere $K_{per}$ a $K_{par}$ = 0.01 pri štatistike 100 miliárd simulácii pre test I.




Obr. 7.8: Pomer energetických spektier 2D F-T a 2D B-p modelu s dt = 5.0 s, $K_0$ = 5 * $10^{18}$ $m^2/s$, rýchlosti V = 400km/h a pomere $K_{per}$ a $K_{par}$ = 0.1 pri štatistike 50 miliárd simulácii pre test II.



Obr. 7.9: Pomer energetických spektier 2D F-T a 2D B-p modelu s dt = 5.0 s, $K_0$ = 5 * $10^{18}$ $m^2/s$, rýchlosti V = 300km/h a pomere $K_{per}$ a $K_{par}$ = 0.01 pri štatistike 50 miliárd simulácii pre test III.



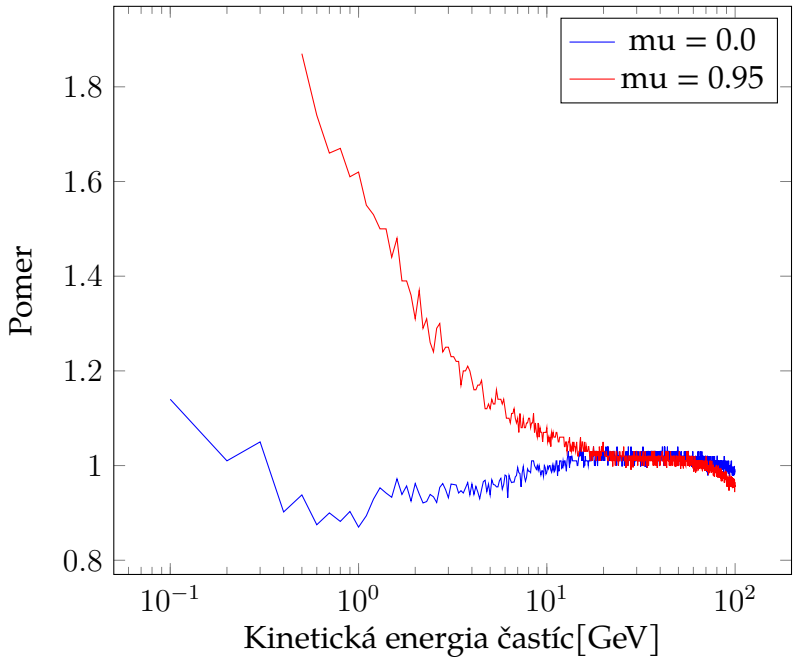

T a 2D B-p modelu sa nachádza na obrázku 7.10, sa preukázalo, že tvar spektra pri mu = 0.0 aj pre mu = 0.95 ostal rovnaký pri pri teste I. a teste III. Kým pomer spektier pre mu = 0.0 vykazuje maximálne 14% rozdiel pri 0.1 GeV, tak pre pomer spektier pre mu = 0.95 klesá pod rozdiel 50% pri 1.5 GeV.

Na obrázku 7.11 sa nachádza pomer energetických spektier GPU implementácie 2D F-T a 2D B-p modelu pre test V. Cieľom test V. a VI. je otestovať maximálnu moduláciu. Pomery energetických spektier z oboch testov sú takmer identické. V pomere spektier z testu VI., nachádzajúcich sa na obrázku 7.12, je jasne viditeľná pulzácia, ktorá sa ale nenachádza v spektre z testu V.. Preto je možné konštatovať, že pulzácia v 2D F-T modeli nastáva pri vysokej modulácii s krátkou dĺžkou kroku dt.

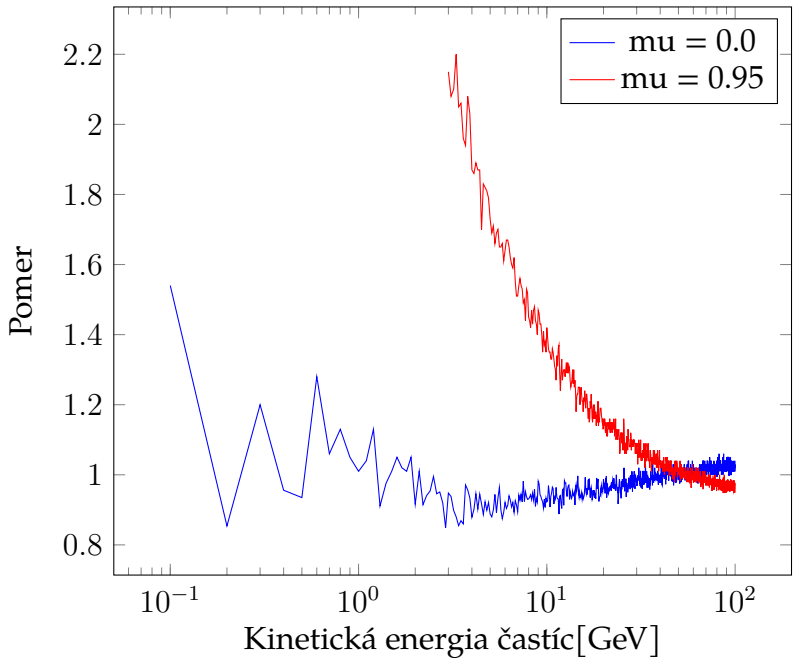
### 7.2.3 Zrýchlenie získané pri GPU implementácii 2D F-p modelu

Vyhodnotenie celkového zrýchlenia sme realizovali pri vstupných parametroch dĺžky kroku dt = 5.0 s, $K_0$ = 5 * $10^{18}$ $m^2/s$, rýchlosti V = 400km/h a pomere $K_{per}$ a $K_{par}$ = 0.01 pri štatistike 100 miliárd simulácii. Pri časovom kroku dt = 5.0 na-

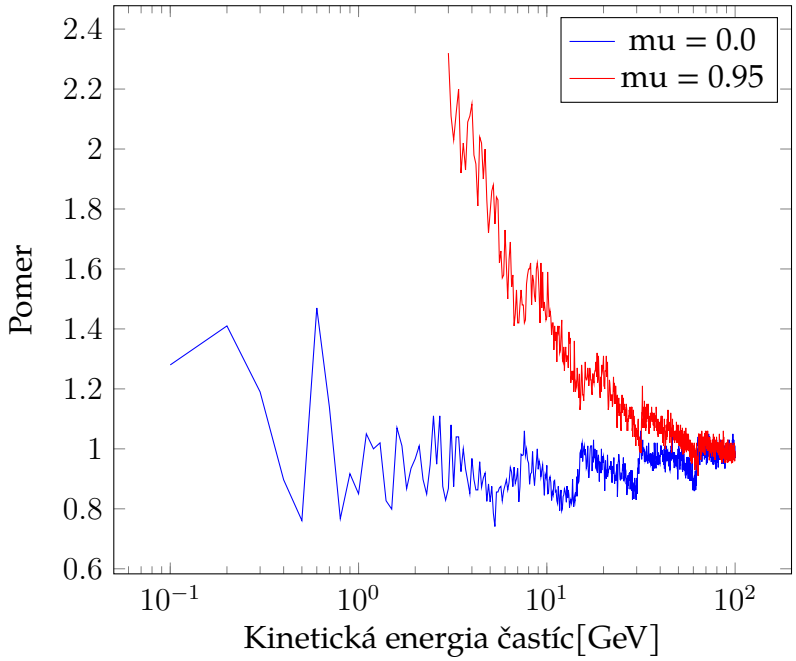
Obr. 7.10: Pomer energetických spektier 2D F-T a 2D B-p modelu s dt = 5.0 s, $K_0$ = 5 * $10^{18}$ $m^2/s$, rýchlosti V = 700km/h a pomere $K_{per}$ a $K_{par}$ = 0.01 pri štatistike 50 miliárd simulácii pre test IV.




Obr. 7.11: Pomer energetických spektier 2D F-T a 2D B-p modelu s dt = 5.0 s, $K_0$ = 1 * $10^{18}$ $m^2/s$, rýchlosti V = 700km/h a pomere $K_{per}$ a $K_{par}$ = 0.01 pri štatistike 50 miliárd simulácii pre test V.



Obr. 7.12: Pomer energetických spektier 2D F-T a 2D B-p modelu s dt = 0.5 s, $K_0$ = 1 * $10^{18}$ $m^2/s$, rýchlosti V = 700km/h a pomere $K_{per}$ a $K_{par}$ = 0.01 pri štatistike 25 miliárd simulácii pre test VI.




stala chyba, kedy začalo spektrum stúpať Táto chyba nie je dôsledkom prepisu, ale vyskytuje sa [11] taktiež v 1D F-p modeli, od ktorého je 2D F-p model odvodený.

Vykonávací čas CPU implementácie 2D F-p modelu pre 100 miliárd simulácii na referenčnom multiprocesorovom systéme bol 150 hodín. Vykonávací čas na systéme s GTX 1080 TI bol pre rovnakú štatistiku 12.82 hodiny. To predstavuje viac ako 11.69-násobné zrýchlenie.

### 7.2.4 Vyhodnotenie presnosti GPU implementácii 2D F-p modelu

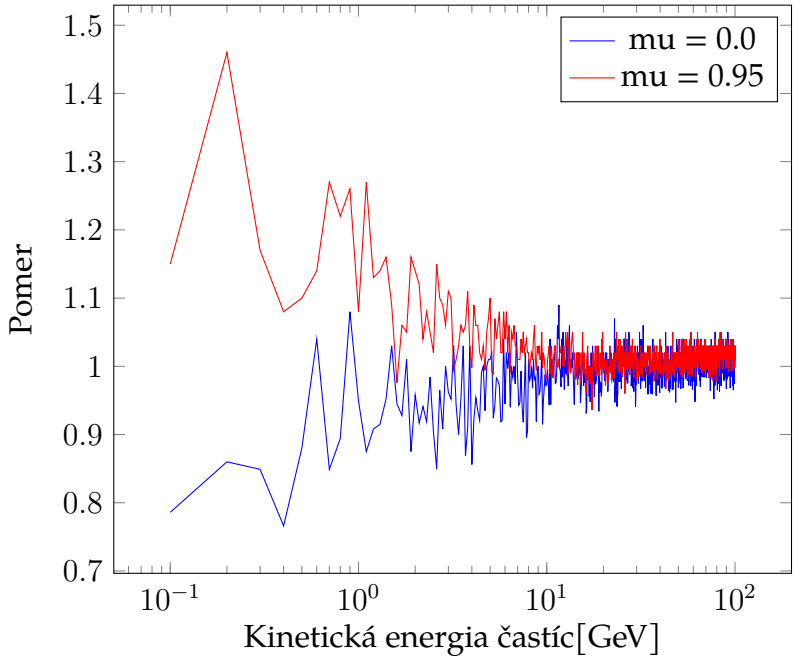
Z hľadiska kvantity získaných údajov sme získali viac údajov, ako v prípade 1D F-p modelu, ale taktiež viac ako v prípade 2D F-T modelu. V tabuľke 7.3 sa nachádza výstupné množstvo z jednotlivých testov pri štatistike 100 miliárd simulácii pre GPU implementáciu 2D F-p modelu. V prípade testu VI. bolo vyprodukovaných až 205.5 GB údajov. Oproti výstupnému množstvu údajov GPU implementácie 2D F-T modelu uvedených v kapitole 7.2.2 má každý test väčší výstup v priemere o 13.36%.

Na obrázku 7.13 sa nachádza pomer energetických spektier 2D F-p a 2D B-p

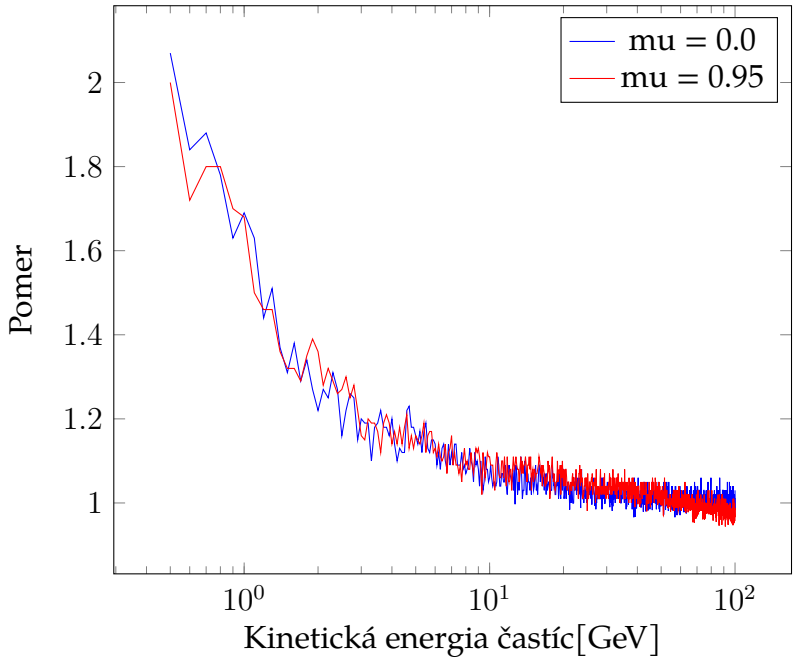
Tabuľka 7.3: Výstupné množstvo údajov z jednotlivých testov pri štatistike 100 miliárd simulácii pre GPU implementáciu 2D F-p modelu

| Označenie testu | Množstvo údajov[GB] |
|---|---|
| I. | 137.4 |
| II. | 33.1 |
| III. | 104.6 |
| IV. | 189.5 |
| V. | 181.3 |
| VI. | 205.5 |

Obr. 7.13: Pomer energetických spektier 2D F-p a 2D B-p modelu s dt = 2.0 s, $K_0$ = 5 * $10^{18}$ $m^2/s$, rýchlosti V = 400km/h a pomere $K_{per}$ a $K_{par}$ = 0.01 pri štatistike 100 miliárd simulácii pre test I.



Obr. 7.14: Pomer energetických spektier 2D F-p a 2D B-p modelu s dt = 2.0 s, $K_0$ = 5 * $10^{18}$ $m^2/s$, rýchlosti V = 400km/h a pomere $K_{per}$ a $K_{par}$ = 0.1 pri štatistike 100 miliárd simulácii pre test II.

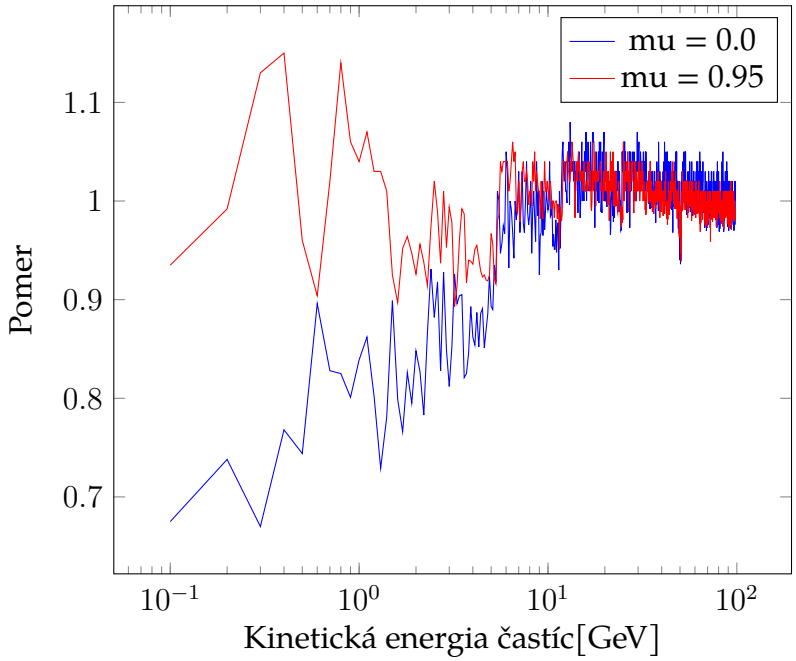



modelov pre test I. Pre mu = 0.0 vidíme rozdiely nad 10% do približne 3.3 GeV. Nad 10 GeV sú rozdiely minimálne, pohybujú sa na úrovni maximálne 4%. Pre mu = 0.95 vidíme odchýlku na nízkych energiách, s najvyššou odchýlkou 46% na 0.2 GeV. Pre energie nad 10 GeV je odchýlka maximálne 2%. Okrem časového kroku dt = 2.0 sme vykonali testovanie aj pre časový krok dt = 5.0. Na vyšší energiách sa objavilo stúpanie energetického spektra, ktoré je chybou modelu. Oproti CPU implementácii sa objavili pulzy v tejto stúpajúcej oblasti od 70 GeV. Oproti referenčnému spektru bol rozdiel až na úrovni $10^{73}$. Pravdepodbnou príčinou problému bolo pretečenie rozsahu jednoduchej presnosti pre jednu resp. viacero hodnôt.

Na obrázku 7.14 sa nachádzajú výsledné pomery energetických spektier GPU implementácii 2D F-p a 2D B-p modelu pre test II.. Oproti testu I. bola veľkosť výsledného súboru len na úrovni 24.09%. Pre mu = 0.0 a mu = 0.95 bola početnosť častíc nižšia o 10-20% na energiách do 5 GeV. Na nízkych energiách do 0.5 GeV je pomer oproti referenčnému B-p spektru viac ako dvojnásobný. Od 10 GeV sa odchýlka od referenčného spektra pohybuje na úrovni najviac 9%.

Pre test III. boli zaznamenaný maximálne 35% rozdiel v prípade mu = 0.0. Pomer výsledných spektier 2D F-p a 2D B-p modelu pre test III. sa nachádza na

Obr. 7.15: Pomer energetických spektier 2D F-p a 2D B-p modelu s dt = 2.0 s, $K_0$ = 5 * $10^{18}$ $m^2/s$, rýchlosti V = 300km/h a pomere $K_{per}$ a $K_{par}$ = 0.01 pri štatistike 100 miliárd simulácii pre test III.



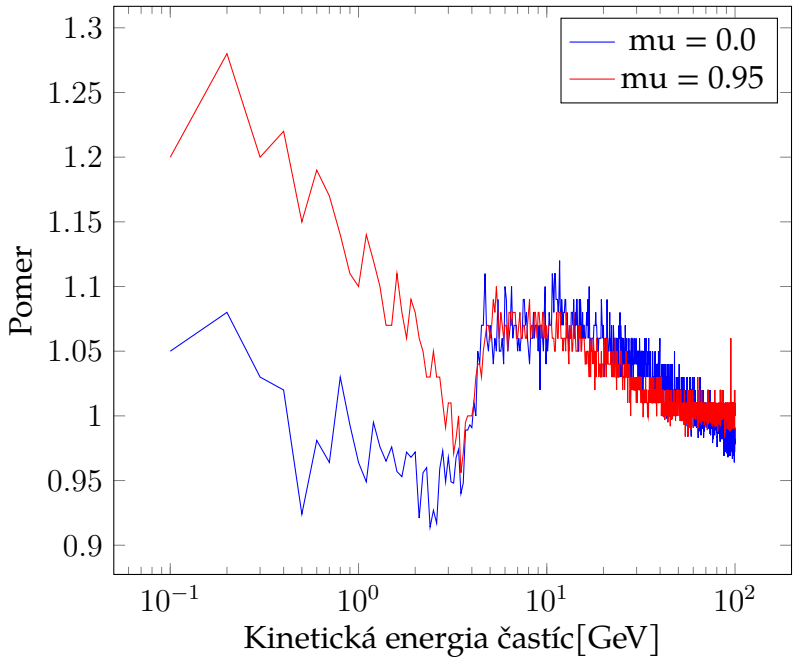

obrázku 7.15. Pre mu = 0.95 sa vyskytuje maximuálny rozdiel na úrovni 15%. Pre energie nad 10 GeV sú rozdiely na úrovni maximálne 5%.

Test IV. pre pomer energetických spektier z GPU implementácii 2D F-p a 2D B-p modelu, ktorý sa nachádza na obrázku 7.16, vykazuje náznaky výskytu pulzu medzi 2-3 GeV pre mu = 0.95. Pre mu = 0.0 sa takýto pulz nenachádza medzi 2-3 GeV. Od 4 GeV do 20 GeV sa pohybuje rozdiel medzi 5-10%. Od 20 GeV klesá rozdiel na maximálne 3% pri vysokých energiách medzi 90-100 GeV.

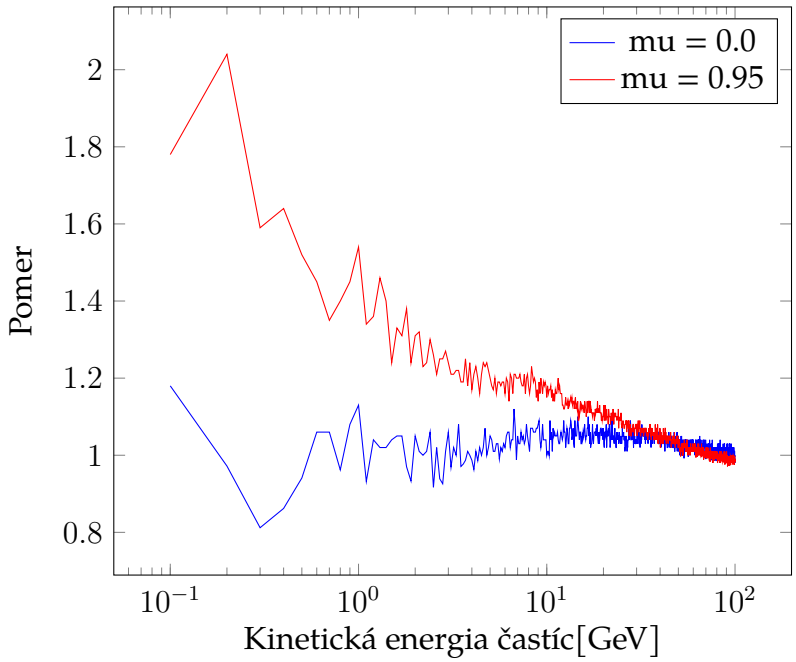
Test V. a VI. mal za cieľ overiť správanie sa spektier pre maximálnu moduláciu. V prípade testu V., ktorý sa nachádza na obrázku 7.17, sú viditeľné rozdiely maximálne na úrovni 9% od 0.5 GeV pre mu = 0.0. Pre mu = 0.95 klesá rozdiel pod 10% až od 20.9 GeV.

V teste VI., ktorého pomer energetických spektier 2D F-p a 2D B-p modelu s časovým krokom dt = 0.5 sa nachádza na obrázku 7.18, bola zaznamenaná pulzácia. Tvar pulzácie je podobný tej, ktorá sa vyskytla pri 2D F-T modeli v test V a VI v kapitole 7.2.2. Pulzácie nedosahujú taký vrchol, ako v prípade pulzácii pri GPU implemenetácii 1D F-p modelu. Pulzácie pri 1D F-p modeli stúpali s vyššou energiou, kým v prípade GPU implemenetácie 2D F-p modelu naopak, klesali.

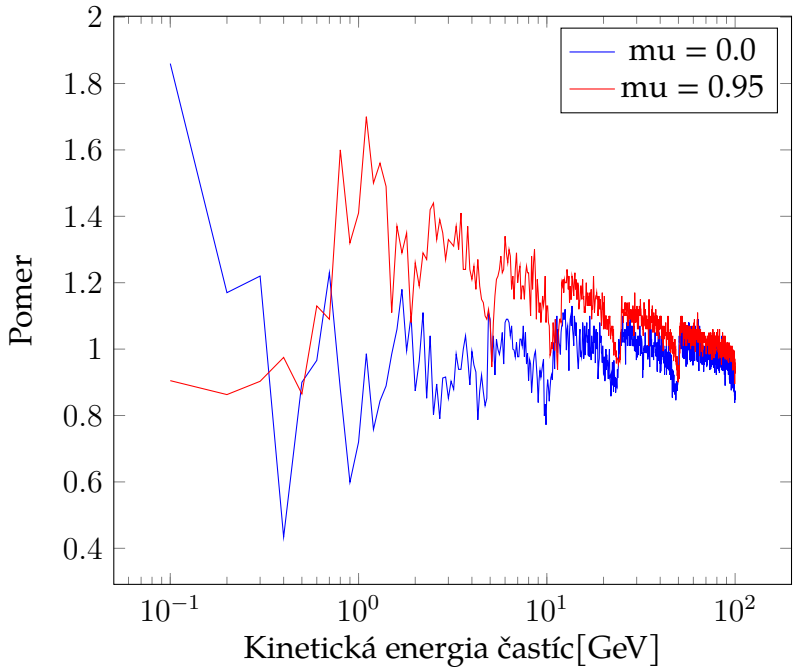
Obr. 7.16: Pomer energetických spektier 2D F-p a 2D B-p modelu s dt = 2.0 s, $K_0$ = 5 * $10^{18}$ $m^2/s$, rýchlosti V = 700km/h a pomere $K_{per}$ a $K_{par}$ = 0.01 pri štatistike 100 miliárd simulácii pre test IV.




Obr. 7.17: Pomer energetických spektier 2D F-p a 2D B-p modelu s dt = 2.0 s, $K_0$ = 1 * $10^{18}$ $m^2/s$, rýchlosti V = 700km/h a pomere $K_{per}$ a $K_{par}$ = 0.01 pri štatistike 100 miliárd simulácii pre test V.



Obr. 7.18: Pomer energetických spektier 2D F-p a 2D B-p modelu s dt = 1.0 s, $K_0$ = 1 * $10^{18}$ $m^2/s$, rýchlosti V = 700km/h a pomere $K_{per}$ a $K_{par}$ = 0.01 pri štatistike 25 miliárd simulácii pre test VI.




## 7.3 2D B-p model

V nasledujúcich podkapitolách sa budeme venovať zrýchleniu, presnosti a analýze spektier, ktoré sme získali z GPU implementácia 2D B-p modelu. Takisto ako pri 2D F-p modeli, mali sme k dispozícii verziu modelu, ktorej fyzika zodpovedá verzii 2.4 2D F-T modelu.

### 7.3.1 Zrýchlenie získané pri GPU implementácii 2D B-p modelu

Taktiež ako pri vyhodnotení 2D F-p a 2D F-T modelu sme sa rozhodli použiť vstupné parametre, dĺžku kroku dt = 5.0s, $K_0$ = 5 * $10^{18}$ $m^2/s$, rýchlosti V = 400km/h a pomere $K_{per}$ a $K_{par}$ = 0.01 pri štatistike 167.7 miliónov simulácii. Ďalším vstupným parametrom je aj oproti 2D F-T alebo 2D F-p mu, pre ktorých hodnoty mu = 0.0 a mu = 0.95 vykonáme simulácie. Oproti modelom typu forward má každá simulácia z backward modelu výsledok v podobe trajektórie spájajúcej registračné miesto v heliosfére a hranicu heliosféry. Preto nie je potrebná tak vysoká štatistika.

Na referenčom multiprocesorovom systéme je výkonávací čas pre ekvivalentnú

štatistiku so vstupným parametrom mu = 0.0 je 62.88 hodín. Pre GPU implementáciu 2D B-p modelu na GTX 1080 TI sme dosiahli vykonávací čas 13.62 minúty. To predstavuje 276.82-násobné zrýchlenie.

Pre mu = 0.95 je vykonávací čas na referenčnom multiprocesorovom systéme 27.95 hodín. Na GTX 1080 TI sme dosiahli vykonávací čas 13.74 minúty. V tomto prípade to predstavuje 121.99-násobné zrýchlenie.

Kvôli nerovnomernému zrýchleniu sme sa rozhodli vykonať test s rovnakými vstupnými parametrami, s výnimkou rýchlosti slnenčného vetra V, pre ktorú sme zvolili V = 700 km/s. Na referenčnom multiprocesorovom systéme pre mu = 0.0 bol dosiahnutý vykonávací čas 74.10 hodiny. Na systéme s GTX 1080 TI bol dosiahnutý vykonávací čas 9.39 minúty. Oproti referenčnému multiprocesorovému systém bolo získané 473.46-násobné zrýchlenie. Pre mu = 0.95 bol dosiahnutý vykonávací čas 12.88 hodiny. Taktiež ako pri teste s V = 400 km/s bolo získané zrýchlenie značne nižšie oproti testu s mu = 0.95. Vykonávací čas na systéme s GTX 1080 TI bol 9.39 minúty, bolo získané 82.29-násobné zrýchlenie oproti referenčnému multiprocesorovému systému.

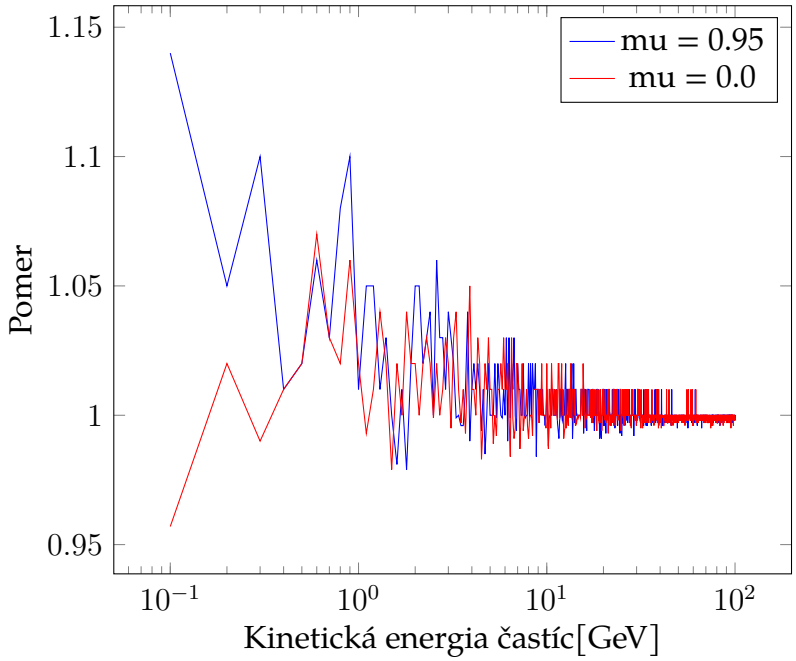
### 7.3.2 Vyhodnotenie presnosti GPU implementácii 2D B-p modelu

Pre GPU implementáciu 2D B-p modelu sme sa rozhodli nemerať kvantitu získaných údajov. Je to najmä z toho dôvodu, že množstvo týchto údajov je limitované, takmer každá simulácia má výsledok, nezaznamenali sme veľmi rozdielnu štatistiku.

Na obrázku 7.19 sa nachádza pomer energetických spektier z GPU a CPU implementácie 2D B-p modelu pre vstupné parametre dĺžky kroku dt = 5.0 s, rýchlosť slnenčného vetra V = 400 km / s. Pre mu = 0.95 bola zaznamenaná 14% miera nepresnosti na 0.1 GeV oproti CPU implementácii 2D B-p modelu. Pre mu = 0.0 bola zaznamenaná maximálne 7% nepresnosť. Príčinou týchto nepresnosti mohla byť aj nízka štatistika referenčného spektra z CPU implemenetácie 2D B-p modelu.

Na obrázku 7.20 sa nachádza pomer výsledných energetických spektier z GPU a CPU implementácie 2D B-p modelu pre rýchlosť slnečného vetra V = 700 km/s. V prípade mu = 0.95 na nízkych energiách je viditeľná maximálne 9% odchýlka na 0.3 GeV a 0.8 GeV. Do 10 GeV sa vyskytovali okrem zmienených odchýliek

Obr. 7.19: Pomer energetických spektier 2D B-p modelu GPU a CPU implementácie s dt = 5.0 s, $K_0 = 5 * 10^{18}$ $m^2/s$, rýchlosti V = 400km/h a pomere $K_{per}$ a $K_{par} = 0.01$ pri štatistike 167 miliónov simulácii



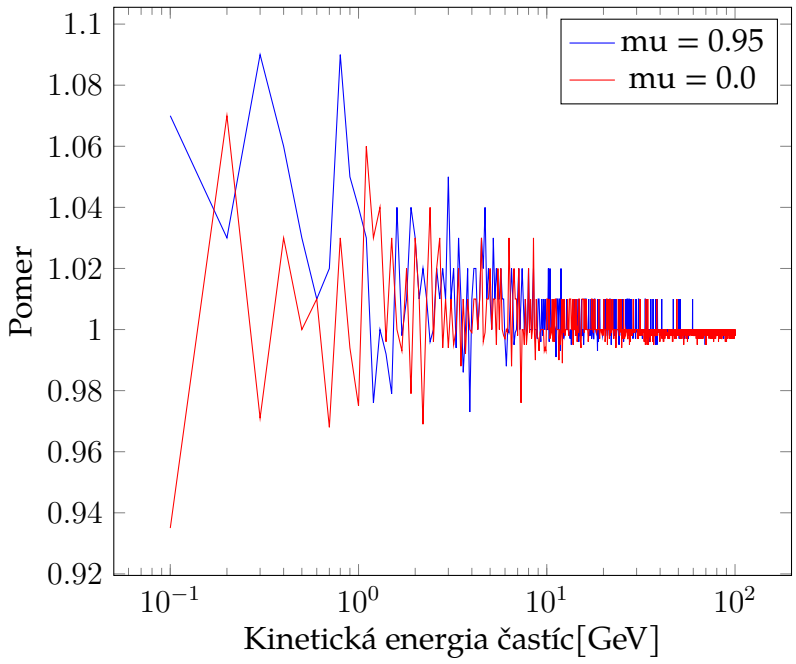

nepresnosti na úrovni 6%. Nad 10 GeV bol zaznamenaný rozdiel len na úrovni maximálne 1%.

Z vykonaných testov vyplýva, že presnosť GPU implementácie 2D B-p modelu bola zachovaná na prijateľnej úrovni. Najväčšie odchýlky sa vyskytovali na nižších energiách. Pre energie vyššie, ako 10 GeV bola odchýlka na úrovni maximálne 1%.

## 7.4 Distribuovaný systém

Pri vyhodnotení distribuovaného systému sa zameriame na dosiahnuté zrýchlenie pri jednotlivých typoch simulácii, so zameraním na backward a forward. Budeme vyhodnocovať aj schopnosť systému reagovať na chyby a následné zotavenie sa z nich.

Obr. 7.20: Pomer energetických spektier 2D B-p modelu GPU a CPU implementácie s dt = 5.0 s, $K_0 = 5 * 10^{18}$ $m^2/s$, rýchlosti V = 700km/h a pomere $K_{per}$ a $K_{par} = 0.01$ pri štatistike 167 miliónov simulácii




### 7.4.1 Vyhodnotenie zrýchlenia na úrovni distribuovaného systému

Vyhodnotenie presnosti bolo realizované na distribuovanom systéme, v ktorom boli zapojené dva uzly distribuovaného systému. Tie obsahovali GPU GTX 1080 TI a RTX 2060. Výkonnostne sa tieto GPU značne odlišujú. Vďaka tejto odlišnosti budú otestované aj distribúcie počtu simulácii pre jednotlivé GPU, ktorých implementácia bola opísaná v kapitole 6.3.

Pre model typu forward sme zvolili 1D F-p model. Ako vstupné parametre sme zvolili časový krok dt = 5.0s, $K_0 = 5 * 10^{18}$ $m^2/s$ a rýchlosť V = 400km/h pre štatistiku 100 miliárd simulácii.

V tabuľke 7.4 sa nachádzajú celkové vykonávacie časy implementácii 1D F-p modelu pre jednotlivé systémy. Oproti jednej GTX 1080 TI sme získali 26.89% nárast výkonu. Ak sa pozrieme na tabuľku 6.2, tak vidíme, že rozdiel vykonávacích časov medzi jednotlivými kartami je len 6 minút, čo predstavuje 2.36% rozdiel. Tento rozdiel je akceptovateľný. V konečnom dôsledku sme s distribuovaným systémom v tejto konfigurácii dosiahli 10.11-násobné zrýchlenie oproti pôvodnému

Tabuľka 7.4: Celkové vykonávacie časy pre implementácie 1D F-p modelu pre 100 miliárd simulácii na jednotlivých systémoch

| Typ systému | Vykonávací čas[hod] |
|---|---|
| Referenčný multiprocesorový systém | 43.30 |
| GTX 1080 TI | 5.61 |
| Distribuovaný systém | 4.28 |

Tabuľka 7.5: Celkové vykonávacie časy pre implementácie 2D F-p modelu pre 100 miliárd simulácii na jednotlivých systémoch

| Typ systému | Vykonávací čas[hod] |
|---|---|
| Referenčný multiprocesorový systém | 150.00 |
| GTX 1080 TI | 12.82 |
| Distribuovaný systém | 10.5 |
| Distribuovaný systém po korekcii | 10.72 |

referenčnému systému.

Pre overenie bol realizovaný výpočet s identickými parametrami aj pre 2D F-p model, s použitím pomeru $K_{par}$ a $K_{per} = 0.01$. Výsledné vykonávacie časy sa nachádzajú v tabuĺke 7.5. Zrýchlenie oproti referenčnému multiprocesorovému systému je až 14.63-násobné. Rozdiel vykonávacích časov medzi GTX 1080 TI a distribuovaným systémom predstavuje 19.89%. Získaný rozdiel je menší ako pri 1D F-p modeli, z čoho vyplýva, že distribúcia simulácii pre jednotlivé uzly nie je optimálna v prípade 2D F-p modelu. Rozdiel medzi vykonávacimi časmi GTX 1080 TI a RTX 2060 v distribuovanom systéme bol 24.04% s tým, že systém s GTX 1080 TI skončil výpočet neskôr. Preto pre výpočet s korekciou distribuovania simulácii bol použitý vzorec uvedený v rovnici 6.1 bez použitia priepustnosti pamäte pre GPU. Preukázalo sa, že výsledok bol o 13.6 minúty horší. Verzia bez korekcie sa ukázala ako o 2.07% rýchlejšia.

Overenie backward modelov bol použitý výpočet s 1D B-p pre 1.6 miliardy simulácii. Výsledné vykonávacie časy pre všetky použité systémy sa nachádzajú v tabuľke 7.6. Zrýchlenie oproti referenčnému multiprocesorovému systému je 102.51-násobné. Rozdiel medzi distribuovaným systémom a GTX 1080 TI je len 13.5%. Simulácie typu backward sú veľmi rýchle oproti simuláciam typu forward.

Tabuľka 7.6: Celkové vykonávacie časy pre implementácie 1D B-p modelu pre 1.6 miliardy simulácii na jednotlivých systémoch

| Typ systému | Vykonávací čas[hod] |
|---|---|
| Referenčný multiprocesorový systém | 145.57 |
| GTX 1080 TI | 1.66 |
| Distribuovaný systém | 1.42 |
| Distribuovaný systém po korekcii | 1.17 |

Rozdiel medzi RTX 2060 a GTX 1080 TI v rámci výpočtu na distribuovanom systéme bol 19.64% s použitím vzorca, ktorý nevyužíva priepustnosť pamäte. Pri prepočte, aký čas vychádza na výpočet jednej iterácie, 16 miliónov simulácii, bol zistený rozdiel na úrovni 9.93%. Distribúcia simulácii v tomto prípade nie je správna. Ako korekciu sme sa rozhodli zmeniť distribúciu simulácii na rovnomernú medzi jednotlivé uzly distribuovaného systému. Korekcia bola správna, vykonávací čas na distribuovanom systéme klesol na 1 hodinu a 17 minút. Predstavuje to zlepšenie o 34.62% oproti GTX 1080 TI.

Efektívnosť delenia v danej konfigurácii distribuovaného systému predstavuje úsporu vykonávacieho času od 24.04% do 34.62% oproti vykonávaniu na GTX 1080 TI.

### 7.4.2 Vyhodnotenie tolerancie chýb distribuovaného systému

Pri vyhodnotení tolerancie chýb distribuovaného systému bol daný dôraz na to, aby výpočty z uzlov distribuovaného systému, ktoré zlyhali, boli dopočítané. Pre účely testovania boli použité tri počítače, ktoré fungovali ako uzol distribuovaného systému. Prvý uzol distribuovaného systému, ktorý používal na výpočty GTX 1080 TI, sme nechali vykonávať výpočty, druhý oznámil hlavnému uzlu zlyhanie a tretí bol vypnutý bez notifikácie hlavného uzla.

Časť simulácie, ktorú mal vypočítať uzol, ktorý oznámil svoje zlyhanie bola hneď zaradená medzi zlyhané simulácie. Po uplynutí 15 minút, ktoré reprezentujú maximálny čas odozvy uzla distribuovaného systému, bola zaradená medzi zlyhané simulácie aj časť simulácie, ktorá mala byť počítaná na uzle, ktorý bol vypnutý bez notifikácie hlavného uzla.

Po dokončení simulácie na prvom uzle bola zahájená časť simulácie, ktorej vý-

počet mal prebiehať na uzle, ktorý oznámil svoje zlyhanie. Následne bola zahájená posledná tretia časť simulácie.

Celkový vykonávací čas bol o 15% dlhší ako vykonanie identického počtu simulácii na GTX 1080 TI. Tento vykonávací čas predstavuje náklady na spustenie simulácii a čas kým boli zahájené. Test preukázal, že implementácia tolerancie chýb je správna a napriek dlhšiemu vykonávaniu bol hlavný uzol schopný správne identifikovať zlyhania na úrovni uzlov distribuovaného systému a priradiť jednotlivé zlyhané časti simulácie dostupným uzlom, ktoré nevykonávali žiadne výpočty.

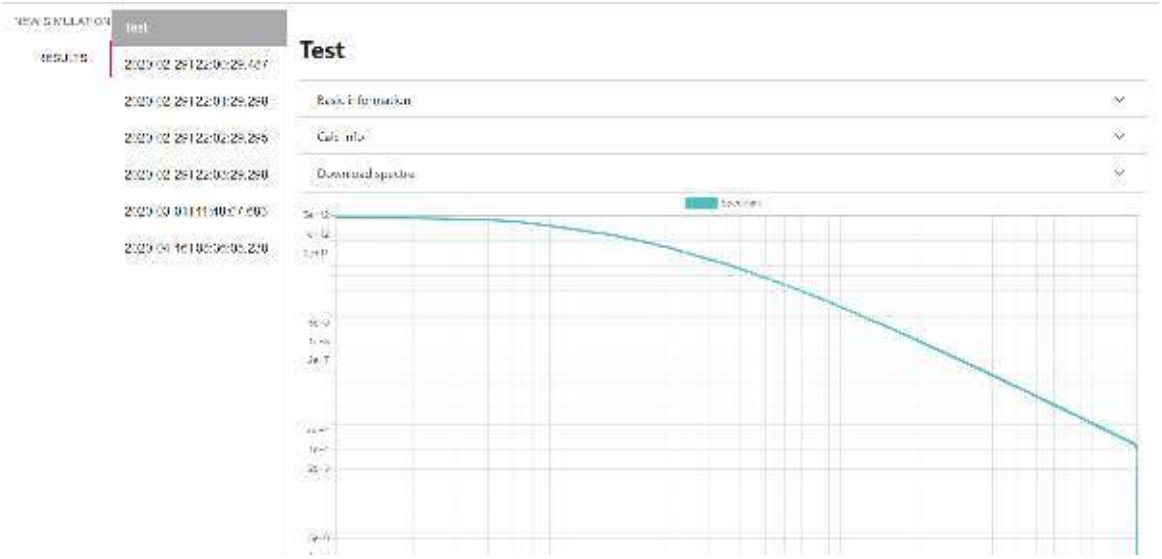
### 7.4.3 Vyhodnotenie webového používateľského rozhrania

Na interakciu s distribuovaným systémom bolo implementované webové používateľské rozhranie. Používateľské rozhranie bolo implementované s dôrazom na zjednodušenie prístupu k spusteniu simulácii a následnému prehľadu zoznamu vypočítaných simulácii.

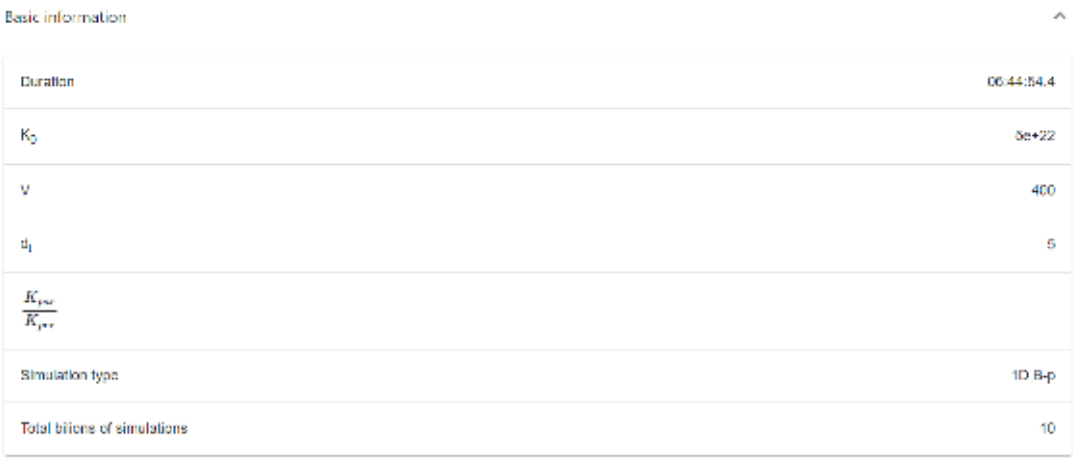
Na obrázku 7.21 je zobrazená implementácia obrazovky pre spustenie nového výpočtu. Na oddelenie jednotlivých simulácii boli použité karty. Každá simulácia má svoju vlastnú kartu, kde sa nachádzajú polia s príslušnými parametrami pre každú simuláciu. Pri prechádzaní kurzorom cez pole formulára sa zobrazí rozsah hodnôt, ktoré je možné zadať. V prípade, ak sú zadané nesprávne hodnoty nie je možné spustiť výpočet. Používateľ má prehľad o tom, aké uzly distribuovaného systému sú k dispozícii a sú voľné k výpočtom.

Na obrázku 7.22 sa nachádza implementácia obrazovky pre zobrazenie výsledkov. Po kliknutí na konkrétnu položku v zozname simulácii sa zobrazia detaily k zvolenej simulácii. Základný prehľad obsahuje parametre výpočtu, trvanie výpočtu, bližší detail je zobrazený na obrázku 7.23. Na obrázku 7.24 sa nachádza implementácia detailných informácii o priebehu výpočtu. Detailné informácie o priebehu výpočtu obsahujú zoznam jednotlivých uzlov distribuovaného systému zapojených do výpočtu. Používateľ má možnosť stiahnuť si výsledné energetické spektrum vo formáte .csv a .dat. Energetické spektrum je zobrazené taktiež v grafe.


Obr. 7.21: Implementácia obrazovky pre spustenie nového výpočtu




Obr. 7.22: Implementácia obrazovky pre zobrazenie výsledkov




Obr. 7.23: Implementácia základných informácii o výpočte


Obr. 7.24: Implementácia detailných informácii o priebehu výpočtu

# 8 Záver

V predloženej práci sme sa venovali trom oblastiam - analýze chyby výskytu takzvanej pulzácie v spektrách GPU implementácie 1D F-p modelu, paralelizácii 2D F-T, 2D F-p a 2D B-p modelov a distribuovanému systému, ktorý umožní distribuovať simulácie na viacero zariadení.

V prípade pulzácii v spektrách GPU implementácie 1D F-p modelu boli navrhnuté a otestované tri hypotézy pôvodu chyby, ktorá spôsobuje problém s pulzáciami, a to: problémy spojené s distribúciou čísel v použitom generátore náhodných čísel, použitie optimalizačných prepínačov a nesprávnu distribúciu počtu simulácii na GPU. Overenie rozdielov medzi dvojitou a jednoduchou presnosťou nebolo realizované z dôvodu časovej náročnosti testu. Pri testovaní sa ani jedna z troch testovaných hypotéz nepotvrdila. Pri odstránení optimalizačného prepínača pulzácia ostala prítomná, bol odstránený len šum na nízkych energiách.

Preto sme pristúpili k analýze rozsahov parametrov, aby sme zistili od ktorých hodnôt vstupných parametrov vznikajú pulzy. Ukázalo sa, že hlavný vplyv má dĺžka kroku dt. Z analýzy spektier vyplynulo, že pulzácie vznikajú pri dĺžke kroku dt $< 2.0$ s.

Pri paralelizácii 2D F-T, 2D F-p a 2D B-p modelov bol kladený hlavný dôraz na zrýchlenie výpočtových časov oproti referenčnému multiprocesorovému systému a zachovaniu prijateľnej presnosti výstupných spektier oproti spektrám z CPU implementácie.

GPU implementácia 2D F-T modelu dosiahla 9.83-násobné zrýchlenie oproti referenčnému multiprocesorovému systému. Pri testovaní maximálnej modulácie sa v modeli objavila pulzácia pri dĺžke kroku dt = 5.0 aj dt = 0.5. Presnosť pri ostatných testoch bola stabilná, pre kosínus heliosférickej košírky mu = 0.0 bol zaznamenaný maximálne 26% rozdiel na 0.1 GeV pri porovnávaní s referenčným 2D B-p spektrom. Pre mu = 0.95 boli zaznamenané odchýlky vyššie ako 50% najmä

na nízkych energiách do 3 GeV.

Kým pri GPU implementácii 1D F-p modelu sme zaznamenali zrýchlenie na úrovni 7.87, pri GPU implementácii 2D F-p modelu sme zaznamenali vykonávací čas 12.82 hodiny pre 100 miliárd simulácii, t.j. 11.69-násobné zrýchlenie oproti referenčnému multiprocesorovému systému. Presnosť oproti 2D B-p modelu sa odlišuje pri energii nad 10 GeV najviac o 10%.

Pri testovaní maximálnej modulácie pri GPU implementácii 2D F-T a 2D F-p modelu bola zaznamenaná pulzácia. To nás vedie k hypotéze, podľa ktorej pulzy vznikajú kvôli nízkym hodnotám $K_0$ a nízkej dĺžke kroku dt.

Presnosť GPU implementácie 2D B-p modelu ostala zachovaná, resp. na prijateľnej úrovni s maximálne 9% odchýlkou na nízkych energiách do 1 GeV. Na energiách medzi 1-10 GeV sa vyskytla maximálne 5% odchýlka oproti referenčnému spektru vypočítanému pomocou CPU implementácie 2D B-p modelu. Na energiách vyšších ako 10 GeV bola zaznamenaná maximálne 1% nepresnosť. Z hľadiska zrýchlenia bolo dosiahnuté namerané najmenšie zrýchlenie na úrovni 82.29 násobku oproti referenčnému multiprocesorovému systému, oproti tomu maximálne namerané zrýchlenie bolo 473.46-násobné.

Získané zrýchlenie na jednej referenčnej grafickej karte pri každom z implementovaných modelov, či už jednorozmerných alebo dvojrozmerných, nás viedlo k návrhu distribuovaného systému určeného pre heterogénne systémy z hľadiska GPU.

Uzly distribuovaného systému sme nasadili na referenčné počítačové systémy s GTX 1080 TI a RTX 2060. Pre test modelov typu forward bol zvolený 1D F-p model. Výsledný vykonávací čas pre 100 miliárd simulácii bol 4 hodiny a 17 minút. To predstavuje 10.11-násobné zrýchlenie oproti referenčnému multiprocesorovému systému. Oproti vykonávaniu na GTX 1080 TI bolo získané o 25% vyššie zrýchlenie. Obdobné testovanie s 2D F-p modelom dosiahlo o 24.04% väčšie zrýchlenie oproti samostatnému vykonávaniu na GTX 1080 TI. Pri testovaní modelov typu backward sa pre GPU implementáciu 2D B-p modelu podarilo dosiahnuť o 34.62% nižší vykonávací čas oproti GTX 1080 TI.

Distribuovaný systém sa preukázal ako odolný voči pádom jednotlivých uzlov distribuovaného systému. Testovali sme distribuovaný systém s troma uzlami distribuovaného systému pri čom dva z nich sme ukončili, jeden s notifikáciou hlavného uzla a druhý bez. Distribuovaný systém následne reagoval korektne a zara-

dil obe časti simulácie medzi zlyhané simulácie. Obe časti boli neskôr vypočítané prvým uzlom distribuovaného systému s 10% oneskorením oproti tej istej karte, keby bol na nej výpočet spustený samostatne.

Používateľské rozhranie v príkazovom riadku bolo zachované a implementácie jednotlivých dvojrozmerných modelov boli doplnené spolu so vstupnými parametrami medzi jednorozmerné modely a ich parametre. Ku distribuovanému systému sme navrhli a implementovali webové používateľské rozhranie, ktoré slúži na spúšťanie simulácii a prezeranie si výsledkov simulácii.

Realizovaná paralelizácia na GPU pre dvojrozmerné modely bola úspešná, priniesla väčšie zrýchlenie, ako bolo očakávané z GPU implementácii jednorozmerných heliosferických modelov. Používateľské rozhranie, v príkazovom riadku pre jednu grafickú kartu alebo webové pre distribuovaný systém, umožní používateľom z radov študentov fyziky na UPJŠ a pracovníkov OKF ÚEF SAV používateľsky jednoduchú interakciu s možnosťami spustenia simulácii pre jednotlivé modely.

V budúcnosti je plánovaný ďalší vývoj nástrojov vypracovaných v rámci diplomovej práce. GPU implementácie heliosferických modelov propagácie kozmického žiarenia v heliosfére budú zahrnuté do projektu, ktorý zverejňuje a sprístupňuje modely kozmického žiarenia verejnosti a fyzikálnej komunite. Plánovaný ďalší rozvoj modelov bol popísaný v projekte RAMA, ktorý bol podaný v rámci PECS výzvy Európskej vesmírnej agentúry v novembri 2019. Plán zahŕňa rozšírenie modelu napríklad o drift častíc v heliosfére.

# Literatúra


[1] Michal Solanik. *Prepis modelov distribúcie kozmického žiarenia v heliosfére do CUDA jazyka*. 2018.

[2] *Testing*. URL: https://docs.nvidia.com/cuda/curand/testing.html#testing (cit. 14. 10. 2019).

[3] PIERRE L'ECUYER a RICHARD SIMARD. „TestU01: AC Library for Empirical Testing of Random Number Generators". In: *ACM Transactions on Mathematical Software* 10 (), s. 1268776–1268777.

[4] *CUDA C Programming Guide*. URL: https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html (cit. 28. 09. 2019).

[5] Mark Harris. *CUDA Pro Tip: Flush Denormals with Confidence*. URL: https://devblogs.nvidia.com/cuda-pro-tip-flush-denormals-confidence/ (cit. 16. 11. 2019).

[6] Blaise Barney et al. „Introduction to parallel computing". In: *Lawrence Livermore National Laboratory* 6.13 (2010), s. 10.

[7] R Du Toit Strauss a Frederic Effenberger. „A hitch-hiker's guide to stochastic differential equations". In: *Space Science Reviews* 212.1-2 (2017), s. 151–192.

[8] Michael J Flynn. „Some computer organizations and their effectiveness". In: *IEEE transactions on computers* 100.9 (1972), s. 948–960.

[9] Andrew S Tanenbaum. *Structured computer organization*. Pearson Education India, 2016.

[10] Andrew S Tanenbaum a Maarten Van Steen. *Distributed systems: principles and paradigms*. Prentice-Hall, 2007.

[11] P Bobik et al. „On the forward-backward-in-time approach for Monte Carlo solution of Parker's transport equation: One-dimensional case". In: *Journal of Geophysical Research: Space Physics* 121.5 (2016), s. 3920–3930.

[12] Pavol Bobik et al. „2D Stochastic Montecarlo to evaluate the modulation of GCR for positive and negative periods". In: *Proc. 21th ECRS 2008 (Košice, Slovakia)* (2009).

[13] Hermann Kopetz a Paulo Verissimo. „Real time and dependability concepts". In: *Distributed systems (2nd Ed.)* ACM Press/Addison-Wesley Publishing Co. 1993, s. 411–446.

[14] Edward Curry. „Message-oriented middleware". In: *Middleware for communications* (2004), s. 1–28.

[15] Jorge E Luzuriaga et al. „A comparative evaluation of AMQP and MQTT protocols over unstable and mobile networks". In: *2015 12th Annual IEEE Consumer Communications and Networking Conference (CCNC)*. IEEE. 2015, s. 931–936.

[16] Tomasz Szydło, Pawel Suder a Jakub Bibro. „Message-oriented communication for ipv6-enabled pervasive devices". In: *Computer Science* 14 (2013).

[17] M Castro et al. „Tsunami-HySEA: a GPU-based model for tsunami early warning systems". In: *Proc XXIV Congress on Differential Equations and Applications, June, Cádiz, Spain*. 2015, s. 8–12.

[18] Vladimir Lončar et al. „OpenMP, OpenMP/MPI, and CUDA/MPI C programs for solving the time-dependent dipolar Gross–Pitaevskii equation". In: *Computer Physics Communications* 209 (2016), s. 190–196.

[19] Miloš Ivanović et al. „Distributed multi-scale muscle simulation in a hybrid MPI–CUDA computational environment". In: *simulation* 92.1 (2016), s. 19–31.

[20] Salvatore Cuomo et al. „Toward a multi-level parallel framework on GPU cluster with PetSC-CUDA for PDE-based Optical Flow computation". In: *Procedia Computer Science* 51 (2015), s. 170–179.

[21] Satish Balay et al. „PETSc users manual". In: (2019).

[22] Zhihua Dong et al. „High-Performance Multi-Mode Ptychography Reconstruction on Distributed GPUs“. In: *2018 New York Scientific Data Summit (NYSDS)*. IEEE. 2018, s. 1–5.

[23] Al Geist et al. „MPI-2: Extending the message-passing interface“. In: *European Conference on Parallel Processing*. Springer. 1996, s. 128–135.

[24] Jonathan Dursi. *HPC is dying, and MPI is killing it*. URL: https://www.dursi.ca/post/hpc-is-dying-and-mpi-is-killing-it.html (cit. 25. 11. 2019).

[25] *Spark Overview*. URL: https://spark.apache.org/docs/latest/ (cit. 25. 11. 2019).

[26] *Chapel Documentation*. URL: https://chapel-lang.org/docs/ (cit. 25. 11. 2019).

[27] Robert C Martin. *Clean code: a handbook of agile software craftsmanship*. Pearson Education, 2009.

[28] Joshua Hursey et al. „The design and implementation of checkpoint/restart process fault tolerance for Open MPI“. In: *2007 IEEE International Parallel and Distributed Processing Symposium*. IEEE. 2007, s. 1–8.

[29] Soulla Louca et al. „MPI-FT: Portable fault tolerance scheme for MPI“. In: *Parallel Processing Letters* 10.04 (2000), s. 371–382.

[30] Nvidia. *NVIDIA/nvidia-docker*. URL: https://github.com/NVIDIA/nvidia-docker/wiki/Installation-(Native-GPU-Support) (cit. 03. 12. 2019).

[31] E. N. Parker. „Dynamics of the Interplanetary Gas and Magnetic Fields.“ In: *Apj* 128 (nov. 1958), s. 664. DOI: 10.1086/146579.

[32] C Pei et al. „A general time-dependent stochastic method for solving Parker's transport equation in spherical coordinates“. In: *Journal of Geophysical Research: Space Physics* 115.A12 (2010).

[33] J Randy Jokipii a J Kota. „The polar heliospheric magnetic field“. In: *Geophysical research letters* 16.1 (1989), s. 1–4.

[34] P. Bobik et al. „Systematic Investigation of Solar Modulation of Galactic Protons for Solar Cycle 23 Using a Monte Carlo Approach with Particle Drift Effects and Latitudinal Dependence“. In: *Apj* 745.2, 132 (feb. 2012), s. 132. DOI: 10.1088/0004-637X/745/2/132. arXiv: 1110.4315 [astro-ph.SR].

[35] Mark Harris. *Using Shared Memory in CUDA C/C*. URL: https://devblogs.nvidia.com/using-shared-memory-cuda-cc/ (cit. 27. 11. 2019).

[36] *Server Documentation*. URL: https://www.rabbitmq.com/admin-guide.html (cit. 03. 12. 2019).

[37] *8.14. JSON Types*. URL: https://www.postgresql.org/docs/9.4/datatype-json.html (cit. 30. 01. 2020).

[38] *Nvidia-smi documentantion*. URL: https://developer.download.nvidia.com/compute/DCGM/docs/nvidia-smi-367.38.pdf (cit. 29. 01. 2020).

[39] Barbara Chapman, Gabriele Jost a Ruud Van Der Pas. *Using OpenMP: portable shared memory parallel programming*. Zv. 10. MIT press, 2008.

[40] Robert S Hanmer. *Patterns for fault tolerant software*. John Wiley & Sons, 2013.

# Zoznam skratiek

**1D B-p** 1D Backward metóda pre riešenie Fokker-Planckovej rovnice v tvare s hybnosťou p.

**1D F-p** 1D Forward metóda pre riešenie Fokker-Planckovej rovnice v tvare s hybnosťou p.

**2D B-p** 2D Backward metóda pre riešenie Fokker-Planckovej rovnice v tvare s hybnosťou p.

**2D F-T** 2D Forward metóda pre riešenie Fokker-Planckovej rovnice v tvare s kinetickou energiou T.

**2D F-p** 2D Forward metóda pre riešenie Fokker-Planckovej rovnice v tvare s hybnosťou p.

**CPU** Central Processing Unit.

**GPU** Graphics Processing Unit.

**RAM** Random Access Memory.

# Zoznam príloh

**Príloha A** Hadvérová konfigurácia zariadení

**Príloha B** Používateľská príručka pre rozhranie v príkazovom riadku

**Príloha C** Systémová príručka pre hlavný uzol

**Príloha D** Systémová príručka pre uzol distribuovaného systému

**Príloha E** Systémová príručka pre používateľské rozhranie

**Príloha F** Nasadenie distribuovaného systému

**Príloha G** CD médium - záverečná práca v elektronickej podobe, zdrojové kódy

# A Hardvérová konfigurácia zariadení

Tabuľka A.1: Hardvérová konfigurácia referenčného multiprocesorového systému

| | |
|---|---|
| CPU | 4 x AMD Opteron(TM) Processor 6274 |
| RAM | 32GB |

Tabuľka A.2: Hardvérová konfigurácia systému s GPU GTX1080TI

| | |
|---|---|
| CPU | Intel(R) Core(TM) i5-7500 CPU @ 3.40GHz |
| RAM | 16GB |
| GPU | Asus Geforce GTX1080TI 11GB |

Tabuľka A.3: Hardvérová konfigurácia systému s GPU RTX2060

| | |
|---|---|
| CPU | Intel(R) Core(TM) i5-8400 CPU @ 2.80GHz |
| RAM | 32GB |
| GPU | Asus Geforce RTX2060 6GB |

# B Používateľská príručka pre rozhranie v príkazovom riadku

Program skompilujeme za pomoci nástroja Make. Musíme mať nastavenú cestu ku nástrojom rámca CUDA.

```
make cosmicCUDA
```

V tabuľke B.2 sú uvedené podporované prepínače. Vylučujú sa len tri prepínače, a to pre spustenie jednotlivých simulácii a zobrazenie pomoci.

Vzorový príkaz pri spustenie simulácie F-p modelu s difúznym koeficientom K v hodnote $10^{18}$ $m^2/s$, rýchlosťou slnečného vetra 300 km/s, dĺžkou kroku dt 50,0 a v množstve 500 miliárd simulácii:

```
./cosmicCUDA -F -K 1e18 -V 300 -d 50.0 -N 500
```

V tabuľke B.1 sú uvedené jednotlivé hraničné hodnoty, ktoré akceptujú prepínače. Ich porušenie bude mať za následok nespustenie danej simulácie s konkrétnymi parametrami.

Tabuľka B.1: Podporované hraničné hodnoty jednotlivých prepínačov

| Prepínač | Minimálna hodnota | Maximálna hodnota |
|---|---|---|
| -d | 0,1 | 1000 |
| -K | $10^{18}$ | $10^{19}$ |
| -V | 100 | 1500 |
| -N | 1 | 100000 |
| -E | 0.01 | 0.1 |
| -M | 0.0 | 1.0 |

Tabuľka B.2: Podporované prepínače a ich účel

| Prepínač | Argument | Význam |
|---|---|---|
| -h | - | zobrazí nápovedu k existujúcim prepínačom |
| -t | - | spustí test pre správnu funkciu GPU |
| -F | - | spustí simuláciu s F-p modelom |
| -B | - | spustí simuláciu s B-p modelom |
| -P | - | spustí simuláciu s 2D F-p modelom |
| -T | - | spustí simuláciu s 2D F-T modelom |
| -O | - | spustí simuláciu s 2D B-p modelom |
| -d | float | nastaví časový krok na dĺžku predanú v argumente |
| -K | float | nastaví difúzny koeficient na hodnotu predanú v argumente(v $m^2/s$) |
| -V | float | nastaví rýchlosť slnečného vetra na hodnotu predanú v argumente (v km/s) |
| -N | int | nastaví počet iterácii v základných jednotkách pre danú simuláciu |
| -E | float | nastaví pomer $K_{par}$ a $K_{per}$ pre danú simuláciu |
| -M | float | nastaví pomer kosínus košírky mu pre danú simuláciu |
| -p | std::string | vytvorí priečinok pre simuláciu so zadaným menom |
| -c | - | Súbory z vyhodnotenia budú v csv formáte |

# C Systémová príručka pre hlavný uzol

Systémová príručka pre hlavný uzol distribuovaného systému obsahuje popis jednotlivých balíkov a tried hlavného uzla distribuovaného systému.

## Package sk.sav.cudaheliomaster.amqp

### public class CudaHelioConsumer

Trieda slúži na prijímanie správ od jednotlivých uzlov distribuovaného systému

**public UUID registerClient(String gpu)**

Slúži na registráciu klienta, ako argument prijíma String vytiahnutý z nástroja nvidia-smi.

**public void updateClient(UUID uuid)**

Slúži na updateovanie stavu uzlu distribuovaného systému.

**public void results(ResultMessage resultMessage)**

Slúži na aktualizovanie simulácie a zaznamenania prípadného úspechu alebo neúspechu.

## Package sk.sav.cudaheliomaster.api

### public class ErrorApiModel

Reprezentuje dátový model, ktorý sa odosiela pri zlyhaní.

### public class HelioApiModel

Reprezentuje dátový model, ktorý prijíma dáta pre spustenie simulácie.

### public class NodeApiModel

Reprezentuje dátový model, ktorý obsahuje informácie o uzle distribuovaného systému.

### public class NodeSimulationApiModel

Reprezentuje dátový model, ktorý obsahuje informácie o prebiehajúcej simulácie na uzle distribuovaného systému.

### public class SimulationApiModel

Reprezentuje dátový model, ktorý obsahuje základné informácie o existujúcej simulácii.

### public class SimulationDetailApiModel

Reprezentuje dátový model, ktorý obsahuje detailné informácie o existujúcej simulácii.

## Package sk.sav.cudaheliomaster.controller

### public class CudaHelioController

Poskytuje API ku spúšťaniu a detailom o simuláciach.

**public ResponseEntity<Void> runNewSimulation(HelioApiModel helioApiModel)**

Slúži na spustenie novej simulácie.

**public ResponseEntity<List<SimulationApiModel» getAllSimulations()**

Slúži na získanie všetkých simulácii, ktoré boli zahájené.

**public ResponseEntity<SimulationDetailApiModel> getSimulationDetail(UUID uuid)**

Slúži na získanie konkrétnej simulácie, ak simulácia nie je nájdená, vráti 400 s ErrorApiModel s detailom o chybe.

**public ResponseEntity<Resource> getSimulationFormat(UUID uuid, Boolean isCsv)**

Slúži na získanie súboru s konkrétnym spektrom. Výsledný súbor je vo formáte .dat alebo .csv.

### public class NodeController

Poskytuje API k prehľadu o uzloch distribuovaného systému.

**public ResponseEntity<List<NodeApiModel» getAllNodes()**

Slúži na získanie informácii o všetkých existujúcich uzloch distribuovaného systému.

## Package sk.sav.cudaheliomaster.domain

Obsahuje triedy, ktoré reprezentujú doménu hlavného uzla distribuovaného systému.

### public class FailedNodeSimulation

Reprezentuje simulácie, resp. časti simulácie, ktoré zlyhali a musia byť znova zahájené.

### public class GPUPerformance implements Serializable

Reprezentuje výkonnosť a využitie jednotlivých GPU.

### public class Node

Reprezentuje uzol distribuovaného systému a základné informácie o ňom.

### public class NodeSimulation

Reprezentuje časť simulácie, ktorá sa vykonáva na konkrétnom uzle distribuovaného systému.

### public class Simulation

Reprezentuje simuláciu, jej základné parametre a stav.

## Package sk.sav.cudaheliomaster.dto

Obsahuje triedy, ktoré slúžia na prenos informácii medzi jednotlivými funkciami.

### public class GPUUsage

Reprezentuje využiteľnosť pre danú GPU.

### public class NodeSimulationSize

Reprezentuje veľkosť priradenú pre daný uzol distribuovaného systému.

### public class PingResults

Reprezentuje výsledky odozvy od jednotlivých uzlov distribuovaného systému.

### public class Result

Reprezentuje výsledky simuálácie.

### public class Spectre

Reprezentuje súbor s menom a dĺžkou súboru.

## Package sk.sav.cudaheliomaster.enumeration

Obsahuje enumeračné typy, ktoré sa vyskytujú v hlavnom uzle distribuovaného systému.

### public enum GPUTypeEnum

Reprezentuje typ podporovaných GPU.

### public enum SimulationType

Reprezentuje typ simulácie.

## Package sk.sav.cudaheliomaster.exception

Obsahuje jednotlivé výnimky, ktoré môžu nastať v hlavnom uzle distribuovaného systému.

### public abstract class AbstractDistributionException extends RuntimeException

Abstraktná trieda, od ktorej sú odvodené ďalšie výnimky.

### public class HelioExceptionHandler

Slúži na zachytávanie výnimiek pre všetky kontroléry.

### public class InvalidInputException extends AbstractDistributionException

Výnimka, ktorá vyjadruje neplatný vstup do hlavného uzla distribuovaného systému.

### public class NotFoundException extends AbstractDistributionException

Výnimka, ktorá vyjadruje nenájdenie hľadaného prvku v hlavnom uzle distribuovaného systému.

# Package sk.sav.cudaheliomaster.mapper

Obsahuje jednotlivé triedy, ktoré zabezpečujú mapovanie medzi entitami.

## public class FailedNodeMapper

Slúži na mapovanie, ktoré súvisí s FailedNodeMapper.

**public FailedNodeSimulation mapSimulationToFailedNodeSimulation(Simulation simulation)**

Slúži na mapovanie Simulation na FailedNodeSimulation.

## public class SimulationMapper

Slúži na mapovanie, ktoré súvisí so Simulation.

**public SimulationDetailApiModel mapSimulationToSimulationDetailApiModel(Simulation simulation)**

Slúži na mapovanie Simulation na SimulationDetailApiModel.

**public Simulation mapFtApiModelToSimulation(HelioApiModel helioApiModel)**

Slúži na mapovanie HelioApiModel na Simulation.

**public HelioApiModel mapSimulationToFTApimodel(Simulation simulation)**

Slúži na mapoavanie Simulation na HelioApiModel.

## public class NodeMapper

Slúži na mapovanie, ktoré súvisí so s doménovou entitou Node.

**public List<NodeApiModel> mapNodeListToNodeApiModelList(List<Node> nodeList)**

Slúži na mapoavanie List<Node> na List<NodeApiModel>.

**public NodeApiModel mapNodeToNodeApiModel(Node node)**

Slúži na mapoavanie Node na NodeApiModel.

### public class NodeSimulationMapper

Slúži na mapovanie, ktoré súvisí so s doménovou entitou NodeSimulation.

**public NodeSimulation mapNodeToNodeSimulation(Node node, int size)**

Slúži na mapovanie Node na NodeSimulation s tým, že size určuje počet simulácii, ktoré majú byť vykonané na danom uzle distribuovaného systému.

**public List<NodeSimulationApiModel> mapNodeSimulationListToApiList(List<NodeSimulation> nodeSimulationList)**

Slúži na mapovanie List<NodeSimulation> na List<NodeSimulationApiModel>.

**public NodeSimulationApiModel mapNodeSimulationToNodeSimulationApiModel(NodeSimulation nodeSimulation)**

Slúži na mapovanie NodeSimulation na NodeSimulationApiModel

### public class PhysicMapper

Slúži na mapovanie, ktoré súvisí so s správami odosielané na jednotlivé uzly distribuovaného systému na spustenie výpočtu.

**public PhysicMessage mapToPhysicMessage(HelioApiModel helioApiModel, UUID nodeSimulationUuid, UUID simulationUuid, int size)**

Slúži na mapovanie na konkrétny typ správy podľa simulácie.

## Package sk.sav.cudaheliomaster.repository

Obsahuje jednotlivé rozhrania, ktoré slúžia na prístup k databáze.

## public interface FailedNodeSimulationRepository extends CrudRepository<FailedNodeSimulation, UUID>

Slúži na prístup do databázy kvôli FailedNodeSimulation.

**List<FailedNodeSimulation> findByNodeSimulation()**

Vráti list všetkých zlyhaných simulácii, ktoré obsahujú časť zlyhanej simulácie.

**List<FailedNodeSimulation> findBySimulation()**

Vráti list všetkých zlyhaných simulácii, ktoré obsahujú nezačatú simuláciu.

## public interface GPUPerformanceRepository extends CrudRepository<GPUPerformance, GPUTypeEnum>

Slúži na prístup do databázy kvôli GPUPerformance.

**Set<GPUPerformance> findAllBySimulationType(SimulationType simulationType)**

Vráti set výkonov GPU k jednotlivým simuláciam.

## public interface NodeRepository extends CrudRepository<Node, UUID>

Slúži na prístup do databázy kvôli Node.

**Set<Node> findAllComputing()**

Vráti set všetkých uzlov distribuovaného systému, ktoré vykonávajú výpočty.

## public interface NodeSimulationRepository extends CrudRepository<NodeSimulation, UUID>

Slúži na prístup do databázy kvôli NodeSimulation.

**Optional<NodeSimulation> findByNode(Node node)**

Vráti NodeSimulation, ktorá prebieha na danom Node.

### public interface SimulationRepository extends CrudRepository< Simulation, UUID>

Slúži na prístup do databázy kvôli Simulation.

## Package sk.sav.cudaheliomaster.service

Obsahuje jednotlivé služby, ktoré poskytuje hlavný uzol distribuovaného systému.

### public class CudaHelioService

Slúži na operácia so simuláciami.

**public void runNewSimulation(HelioApiModel helioApiModel)**

Slúži na spustenie výpočtu na distribuovanom systéme, resp. zaradenie do fronty na spracovanie.

**public NodeSimulation launchNewNodeSimulation(Simulation simulation, Node node, int size)**

Zaháji simuláciu na konkrétnom uzle distribuvaného systému.

**public List<SimulationApiModel> getAllSimulations()**

Vráti zoznam všetkých existujúcich simulácii v systéme.

**public SimulationDetailApiModel getSimulationDetail(UUID uuid)**

Vráti detailný prehľad o simulácii s daným UUID.

**public Spectre getSimulationSpectre(UUID uuid, Boolean isCsv)**

Slúži na vytvorenie spektra a jeho zápis do súboru.

### public class FailedSimulationService

Slúži na operácie nad FailedSimulation.

**public void redistributeSimulations()**

Slúži na redistribúciu simulácii, ktoré zlyhali. Funkcia sa púšťa každých 60 sekúnd.

### public class NodeService

Slúži na operácie nad jednotlivými uzlami distribuovaného systému.

**public void setInactiveNodes()**

Označí za neaktívne uzly, pri ktorých bol označený varovný príznak za pravdivý.

**public PingResults pingAllNodesForAvailability()**

Slúži na vykonanie pingu pre všetky uzly distribuvaného systému.

**public void setNodeInactive(PingResults pingResults)**

Označí za neaktívny konkrétne uzly, ktoré sa nachádzajú v parametre PingResults.

**public List<NodeApiModel> getAllNodes()**

Vráti zoznam všetkých existujúcich uzlov distribuovaného systému.

### public class SizeService

Slúži na vykonanie určenie počtu simulácii pre jednotlivé uzly distribuovaného systému.

**public Map<Node, Integer> calculateSize(Set<Node> availableNodes, SimulationType simulationType, Integer size)**

Slúži na vykonanie určenie počtu simulácii pre jednotlivé uzly distribuovaného systému pre danú veľkosť size.

## Package sk.sav.cudaheliomaster.util

Obsahuje pomocné funkcie pre funkcionalitu v hlavnom uzle distribuovaného systému.

### public class GPUUtils

Obsauje pomocné funkcie pre operácie spojené s GPU.

**public static GPUTypeEnum getGPUType(String gpu)**

Na základe vstupného reťazca vráti typ GPU.

**public static String getFilename(Simulation simulation)**

Vyextrahuje zo simulácie meno simulácie, ak nie, vytvorí ho z parametrov.

## Package sk.sav.cudaheliomaster.validator

Obsahuje triedy, ktoré slúžia pre validovanie údajov pre simuláciu.

### public interface HelioValidator

**boolean isValid(HelioApiModel helioApiModel)**

Slúži na validovanie vstupného modelu.

## Package sk.sav.cudaheliomaster.validator.model

Obsahuje triedy, ktoré slúžia pre validovanie údajov pre konkrétne typy simulácii.

### public interface ModelValidator

**boolean validate(HelioApiModel helioApiModel)**

Slúži na validovanie na úrovni konkrétneho typu simulácie.

# D Systémová príručka pre uzol distribuovaného systému

Systémová príručka pre uzol distribuovaného systému obsahuje popis jednotlivých balíkov a tried uzla distribuovaného systému.

## Package sk.sav.cudahelioclient.consumer

Obsahuje jednotlivé triedy, ktoré slúžia na komunikáciu s hlavným uzlom distribuovaného systému.

### public class RegisterClient

Slúži na registráciu uzla distribuovaného systému v hlavnom uzle.

**public void register**()

Načíta už existujúce UUID uzlu distribuovaného systému alebo zaregistruje ho nanovo a vytvorí potrebné fronty.

## Package sk.sav.cudahelioclient.consumer.callback

Obsahuje triedy, ktoré implementujú rozhranie DeliverCallback, ktoré umožňuje reagovať na správu z konkrétnej fronty.

### public class CommunicationCallback implements DeliverCallback

Umožňuje reagovať na správu o spustenie nového výpočtu a zaháji nový výpočet.

### public class PingCallback implements DeliverCallback

Umožňuje reagovať na správu o ping, odosiela naspäť dostupnosť systému.

## Package sk.sav.cudaheliocclient.service

Obsahuje jednotlivé služby, ktoré poskytujú prístup k základným funkciám uzla distribuovaného systému.

### public class CudaProcessService

Obsahuje funkcie, ktoré slúžia k spusteniu procesu so simuláciou.

**public ProcessBuilder buildCudaProcess(PhysicMessage physicMessage)**

Slúži na pripravenie parametrov pre proces so simuláciou na základe obdržaných parametrov zo správy.

### public class HeartBeatService

Obsahuje funkcie, ktorá slúžia k periodickému posielaniu správ na hlavný uzol distribuovaného systému.

**public void updateStatus()**

Každé dve minúty posiela dostupnosť na hlavný uzol distribuovaného systému.

## Package sk.sav.cudaheliocclient.state

### public class Availability

Reprezentuje stav uzla distribuovaného systému - t.j. či je spustená alebo nie je spustená simulácia.

# Package sk.sav.cudahelioclient.utils

## public class GPUUtils

**public String getGPU()**

Vráti názov grafickej karty z nástroja nvidia-smi.

**public String getSimulationDirectoryName(PhysicMessage physicMessage)**

Vráti názov priečinku so simuláciami podľa typu správy.

# E Systémová príručka pre používateľské rozhranie

Systémová príručka obsahuje popis jednotlivých komponentov, ktoré sa nachádzajú vo webovom rozhraní, ktoré je určené pre hlavný uzol distribuovaného systému.

### Dashboard

Slúži na základné zobrazenie - zapuzdruje kompoentn pre spustenie výpočtov, zobrazenie výsledkov a obsahuje stavy pre základné ovládanie upozorňovacích prvkov.

### NewSimulation

Zapuzdruje spustenie jednotlivých druhov simulácii a stav uzlov distribuovaného systému.

### NodeStatus

Slúži na zobrazenie stavu existujúcich uzlov distribuovaného systému, ktoré sú registrované v hlavnom uzle distribuovaného systému.

### OneDimensionBpSimulation

Slúži na spustenie simulácie 1D B-p modelu.

### OneDimensionFpSimulation

Slúži na spustenie simulácie 1D F-p modelu.

### TwoDimensionBpSimulation

Slúži na spustenie simulácie 2D B-p modelu.

### TwoDimensionFpSimulation

Slúži na spustenie simulácie 2D F-p modelu.

### TwoDimensionFTSimulation

Slúži na spustenie simulácie 2D F-T modelu.

### Results

Zapuzdruje komponenty na zobrazenie dostupných simulácii, ich priebehu a výsledkov.

### ListOfResults

Zobrazí všetky existujúce simulácie v distribuovanom systéme.

### Detail

Zapuzdruje jednotlivé komponenty, ktoré zobrazujú detaily o simulácii. Zobrazuje aktuálny stav energetického spektra simulácia.

### BasicInformation

Zobrazuje základné parametre výpočtu a aktuálne trvanie.

### CalcInformation

Zobrazuje podrobnosti o simulácii, ktoré uzly a v akom množstve sa podieľajú na simulácii.

### DownloadSpectre

Slúži na stiahnutie spektra v .dat alebo .csv formáte.

# F Nasadenie distribuovaného systému

Všetky potrebné Dockerfile a docker-compose sa nachádzajú v prílohe na priloženom CD médiu.

Dockerfile pre uzol distribuovaného systému s výpočtovou schopnosťou (angl. compute capabality) 6.1 - t.j. pre architektúru Pascal (pre Turing je paramter turing v make cosmicCUDA rovný 1):

```
FROM maven:3.6.3-openjdk-11-slim as build

WORKDIR /app
COPY cudamodels cudamodels
COPY cudahelioclient cudahelioclient

WORKDIR /app/cudamodels

RUN mvn install -DskipTests

WORKDIR /app/cudahelioclient

RUN mvn package -DskipTests

FROM nvidia/cuda:10.1-devel

WORKDIR /
COPY --from=build
    /app/cudahelioclient/cudahelioclient-0.0.1-SNAPSHOT.jar /
COPY . /
RUN apt update
```

```
RUN apt install -y openjdk-11-jre && apt clean
RUN make cosmicCUDA turing=0

ENTRYPOINT ["java", "-jar", "-Dspring.profiles.active=prod",
    "/cudahelioclient-0.0.1-SNAPSHOT.jar"]
```

Pre hlavný uzol distribuovaného systému má Dockerfile nasledujúce znenie:

```
FROM maven:3.6.3-openjdk-11-slim as build

WORKDIR /app
COPY cudamodels cudamodels
COPY dist dist

WORKDIR /app/cudamodels

RUN mvn install -DskipTests

WORKDIR /app/dist

RUN mvn package -DskipTests

FROM openjdk:8-jdk-alpine3.9

WORKDIR /
COPY --from=build
    /app/dist/target/cuda-helio-master-0.0.1-SNAPSHOT.jar /
EXPOSE 8080

ENTRYPOINT ["java", "-jar", "-Dspring.profiles.active=prod",
    "/cuda-helio-master-0.0.1-SNAPSHOT.jar"]
```

Dockerfile pre webové rozhranie má nasledujúce znenie:

```
FROM tiangolo/node-frontend:10 as build-stage
```

```
WORKDIR /app

COPY package*.json /app/

RUN npm install --silent

COPY ./ /app/

RUN npm run build --silent

FROM nginx:1.15
ENV URL_MASTER_BE=$URL_MASTER_BE
COPY --from=build-stage /app/build/ /usr/share/nginx/html
COPY --from=build-stage /nginx.conf /etc/nginx/conf.d/default.conf
```

Pre časť, ktorú je možné zapuzdriť s hlavným uzlom distribuovaného systému, má docker-compose nasledujúce znenie:

```
version: '3'
services:
  master:
    build: "./master/"
    ports:
    - "8080:8080"
  frontend:
    build: "./frontend/"
    ports:
    - "80:80"
    environment:
        URL_MASTER_BE: master
  rabbitmq:
    image: "rabbitmq"
    ports:
      - "5672:5672"
  postgres:
    image: "postgres"
```