VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
DEPARTMENT OF COMPUTER SYSTEMS

AKCELERACE ULTRAZVUKOVÉ NEUROSTIMULACE POMOCÍ ARITMETIKY SE SNÍŽENOU PŘESNOSTÍ

ACCELERATION OF ULTRASOUND NEUROSTIMULATION USING MIXED-PRECISION ARITHMETIC

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. RADEK DUCHOŇ
AUTHOR

VEDOUCÍ PRÁCE — doc. Ing. JIŘÍ JAROŠ, Ph.D.
SUPERVISOR

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání diplomové práce

144975

Ústav:	Ústav počítačových systémů (UPSY)
Student:	**Duchoň Radek, Bc.**
Program:	Informační technologie a umělá inteligence
Specializace:	Kybernetická bezpečnost
Název:	**Akcelerace ultrazvukové neurostimulace pomocí aritmetiky se sníženou přesností.**
Kategorie:	Paralelní a distribuované výpočty
Akademický rok:	2022/23

Zadání:

1. Seznamte s architekturou výpočetních systémů založených na grafických kartách Nvidia a softwarové platformě CUDA.
2. Prostudujte aktuální implementace a omezení výpočtů se sníženou přesností na grafických kartách Nvidia.
3. Prostudujte současnou implementaci akustického balíku k-Wave a identifikujte datové struktury, vhodné pro redukci přesnosti, např. parametry materiálu.
4. Navrhněme prototyp aplikace a ověřte přesnost výpočtu na jednoduchých úlohách.
5. Transformujte vybrané datové struktury a výpočty nad nimi do nižší přesnosti. Integrujte dané řešení do balíku k-Wave.
6. Na sadě testovacích úloh z oblasti ultrazvukové neurostimulace ověřte dosažené zrychlení a přesnost výpočtu.
7. Zhodnoťte dosažené výsledky a diskutujte přínos práce pro další směrování vývoje balíku k-Wave.

Literatura:

- Dle pokynů vedoucího.

Při obhajobě semestrální části projektu je požadováno:

- Splnění bodů 1-4 zadání.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce:	**Jaroš Jiří, doc. Ing., Ph.D.**
Vedoucí ústavu:	Sekanina Lukáš, prof. Ing., Ph.D.
Datum zadání:	1.11.2022
Termín pro odevzdání:	17.5.2023
Datum schválení:	31.10.2022

Abstrakt

K-Wave je nástroj pro akustickou a ultrazvukovou simulaci s otevřeným zdrojovým kódem. Aktuální dostupné implementace jsou napsány v jazycích C++ a Matlab. Cílem této diplomové práce je akcelerovat existující implementaci ultrazvukové simulace pomocí výpočtů s nižší přesností na grafických kartách Nvidia za využití softwarové platformy CUDA. Dalším přínosem této práce by měla být snížená paměťová náročnost, což umožní provádění větších simulací. Snížená přesnost však nesmí vzhledem k využití například pro neurostimulaci mozku příliš narušit výsledky jako celek. Důležité je proto identifikovat vhodné veličiny, které lze uložit v nižší přesnosti. V této práci budou analyzovány možné přístupy a jejich efektivita při využití nižší přesnosti. Dále pak bude proveden návrh řešení, jehož částí bude identifikace potenciálních veličin pro redukci. Na to bude navazovat specifikace docílené implementace a její testování. Závěr se bude věnovat zhodnocení řešení na základě dosažených výsledků z testování.

Abstract

K-Wave is an open source tool for acoustic and ultrasound simulation. Current available implementations are written in C++ and Matlab. The aim of this thesis is to accelerate the existing implementation of ultrasound simulation by means of lower precision calculations on Nvidia graphics cards using the CUDA software platform. Another benefit of this work should be a reduced memory requirement, which will enable larger simulations to be performed. However, due to the use, for example, for neurostimulation of the brain, the reduced accuracy must not disturb the results as a whole too much. It is therefore important to identify suitable quantities that can be stored in lower precision. In this work, possible approaches and their effectiveness in utilizing lower precision will be analyzed. Furthermore, a solution proposal will be made, which will include identifying potential variables for reduction. This will be followed by specifying the achieved implementation and its testing. The conclusion will focus on evaluating the solution based on the results obtained from the testing.

Klíčová slova

CUDA, C++, Nvidia, GPU, K-Wave, HPC, Akcelerace, Ultrazvuk, Akustické vlnění, Neurostimulace, Poloviční přesnost, Snížená přesnost

Keywords

CUDA, C++, Nvidia, GPU, K-Wave, HPC, Acceleration, Ultrasound, Acoustic waves, Neurostimulation, Half precision, Reduced precision

Citace

DUCHOŇ, Radek. *Akcelerace ultrazvukové neurostimulace pomocí aritmetiky se sníženou přesností.* Brno, 2023. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce doc. Ing. Jiří Jaroš, Ph.D.

Akcelerace ultrazvukové neurostimulace pomocí aritmetiky se sníženou přesností

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana docenta Jiřího Jaroše. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

. . . . . .
Radek Duchoň
17. května 2023

Poděkování

Rád bych zde poděkoval svému vedoucímu práce, panu docentu Jiřímu Jarošovi za odborné vedení této práce, konzultace a podnětné návrhy k řešení. Tato práce byla podpořena Ministerstvem školství, mládeže a tělovýchovy České republiky prostřednictvím e-INFRA CZ (ID:90140).

Obsah

1 Úvod **3**

2 Výpočty se sníženou přesností **5**
2.1 Datové typy 5
2.1.1 Float 5
2.1.2 Half 6
2.1.3 Brain float 6
2.1.4 8bitové floaty 7
2.2 Potenciál zrychlení a úvod do optimalizace 8

3 Testování výkonu v jednoduchých aplikacích **9**
3.1 Kopírování dat 9
3.2 Redukce 12
3.3 Skalární součin 15
3.4 Suma vektorů 18
3.5 Násobení matic 20

4 Analýza programu k-Wave a návrh řešení **22**
4.1 Analýza k-Wave a návrh řešení 22
4.2 Analýza vhodnosti dat pro konverzi do nižší přesnosti 23

5 Implementace **28**
5.1 Přepínač pro redukovanou přesnost 28
5.2 Třídy pro matice dat s redukovanou přesností 29
5.3 Gettery, kernely a související funkce 30
5.4 Redukované veličiny a související úpravy 34

6 Testování a zhodnocení dosažených výsledků **38**
6.1 Dosažené zrychlení 38
6.2 Paměťová úspora 42
6.3 Přesnost výsledků 45

7 Závěr **57**

Literatura **58**

Seznam obrázků

2.1 Porovnání datových typů 7

3.1 Graf rychlosti kopírování dat dle velikosti datového typu - Nvidia A100 . . 10
3.2 Graf zrychlení kopírování dat dle velikosti datového typu - Nvidia A100 . . 10
3.3 Graf rychlosti kopírování dat dle velikosti datového typu - Nvidia V100 . . 11
3.4 Graf zrychlení kopírování dat dle velikosti datového typu - Nvidia V100 . . 11
3.5 Graf Zrychlení výpočtu redukce - Nvidia A100 14
3.6 Graf Zrychlení výpočtu redukce - Nvidia V100 14
3.7 Graf Zrychlení výpočtu skalárního součinu - Nvidia A100 17
3.8 Graf Zrychlení výpočtu skalárního součinu - Nvidia V100 17
3.9 Graf Zrychlení výpočtu sumy vektorů - Nvidia A100 19
3.10 Graf Zrychlení výpočtu sumy vektorů - Nvidia V100 19
3.11 Graf Zrychlení výpočtu násobení matic - Nvidia A100 21
3.12 Graf Zrychlení výpočtu násobení matic - Nvidia V100 21

6.1 Porovnání využití paměti 44
6.2 Vrstva s ohniskem tlaku - PH1-BM7-SC1 46
6.3 Odchylka vrstvy s ohniskem - Minimální redukce - PH1-BM7-SC1 46
6.4 Odchylka vrstvy s ohniskem - Střední redukce - PH1-BM7-SC1 47
6.5 Odchylka vrstvy s ohniskem - Maximální redukce - PH1-BM7-SC1 47
6.6 Grafy normalizovaných odchylek vůči ohnisku - PH1-BM7-SC1 48
6.7 Vrstva s ohniskem tlaku - PH1-BM8-SC1 50
6.8 Odchylka vrstvy s ohniskem - Minimální redukce - PH1-BM8-SC1 50
6.9 Odchylka vrstvy s ohniskem - Střední redukce - PH1-BM8-SC1 51
6.10 Odchylka vrstvy s ohniskem - Maximální redukce - PH1-BM8-SC1 51
6.11 Grafy normalizovaných odchylek vůči ohnisku - PH1-BM8-SC1 52
6.12 Vrstva s ohniskem tlaku - PH1-BM9-SC1 54
6.13 Odchylka vrstvy s ohniskem - Minimální redukce - PH1-BM9-SC1 54
6.14 Odchylka vrstvy s ohniskem - Střední redukce - PH1-BM9-SC1 55
6.15 Odchylka vrstvy s ohniskem - Maximální redukce - PH1-BM9-SC1 55
6.16 Grafy normalizovaných odchylek vůči ohnisku - PH1-BM9-SC1 56

Kapitola 1

Úvod

Tato diplomová práce se věnuje akceleraci simulace šíření akustického tlaku při využití ultrazvuku za pomoci softwarového balíku k-Wave [29]. Tento nástroj má aktuálně více implementací, v této práci se budu věnovat primárně verzi napsané v C++ [30] využívající platformu CUDA [11] pro optimalizované výpočty na grafických kartách Nvidia.

Studium šíření akustického tlaku má v dnešní době řadu lékařských využití. Je díky němu možno například zacílit na konkrétní oblasti mozku a využít tak ultrazvuk pro jejich bezpečnou a neinvazivní neurostimulaci [10]. Nabízí tak možnou alternativu k neuromodulačním metodám jako optogenetika, chemogenetika a magnetická stimulace [23].

Ultrazvuk vlivem absorpce energie také generuje v materiálu teplo. Dalším možným využitím by tak mohlo být indukování dostatečného tepla pro zničení konkrétních buněk rakoviny či potenciálně rakovinotvorných buněk [18]. Právě toto využití přitom nemusí být striktně omezeno na oblast mozku. Lze jej aplikovat také na jiné části těla, které mohou mít řadu jiných problémů, kvůli kterým není vhodné nebo technicky možné se k nim jednoduše dostat fyzicky s cílem vyříznutí nádoru. Ve srovnání s metodami jako je například chemoterapie a radioterapie je tato metoda neinvazivní a lze ji přesněji zaměřit na konkrétní místa. Využití ultrazvuku proto ve srovnání s nimi nemusí trpět na tak velké množství nežádoucích účinků. Mezi typické příznaky těchto léčeb patří například zvýšená únava, bolesti hlavy, ztráta vlasů, poruchy trávení, ale i závažnější problémy, jako je třeba anémie a neutropenie, které mohou vést k dalším závažným zdravotním potížím [7, 27]. Alternativně lze ultrazvuk využít také terapeuticky k mírnění bolesti a senzorických poruch spojených s následky chemoterapie [6].

Akcelerace má být v této práci docíleno za pomoci výpočtů se sníženou přesností. Touto metodou by mělo být docíleno zrychlení primárně, jelikož se sníží objem dat. Klesne tedy nutný počet přístupů do relativně pomalé globální paměti grafických karet. Díky tomu také klesne paměťová náročnost a mělo by být umožněno provádění větších simulací s většími objemy dat. To je velkým přínosem, jelikož paměť grafických karet je ve srovnání s dostupnou pamětí při práci na procesorech zásadně limitována. Další vliv na zkrácení doby výpočtů může mít kratší doba provedení instrukcí určených pro nižší přesnost výpočtů a případná možnost jejich paralelního provádění. Úspěšné zkrácení doby výpočtů by mimo jiné mělo také pozitivní vliv na spotřebu energie. Z tohoto pohledu se tedy jedná také o způsob dlouhodobých ekonomických úspor s kladným enviromentálním dopadem, tzv. green computing má vliv například na snižování uhlíkových emisí [21].

Tato práce se bude v kapitole 2 nejprve věnovat specifikacím datových typů se sníženou přesností. Konkrétně budou probrána jejich omezení a využitelnost. U toho budou také

uvedeny příklady pro snadnější představení konkrétních rozdílů v přesnosti mezi různými typy navzájem.

Následovat bude kapitola 3, ve které se budu zabývat testováním jednoduchých kódů využívajících tyto typy se sníženou přesností, ale i standardní datové typy. Účelem těchto testů je praktické ověření teoretických možností, zrychlení a omezení při práci s daty uloženými ve snížené přesnosti. Zároveň se bude jednat o seznámení se správným způsobem práce s danými typy pro zajištění co nejideálnější efektivity prováděných výpočtů a čekání na paměť.

Kapitola 4 se bude posléze věnovat analýze existující implementace softwarového balíku k-Wave ve verzi využívající platformu CUDA. Bude zde vytvořen návrh řešení pro zakomponování výpočtů s redukovanou přesností. Poslední sekce této kapitoly se poté bude věnovat detailní analýze matic všech veličin, které se v programu k-Wave nacházejí a možnostem jejich převodu do nižší přesnosti. Stejně tak bude vysvětleno, pro které veličiny se z jakých důvodů nemá smysl o redukci přesnosti pokoušet.

V kapitole 5 bude následně popsána reálná implementace práce. Důraz bude na zdůvodnění zvolené implementace a na zhodnocení splnění předešlého návrhu, případně vysvětlení, proč se některé části nebyly implementovány dle návrhu.

Následující kapitola 6 bude posléze věnována otestování vytvořené implementace. Zhodnocena bude celková časová úspora, dosažené zrychlení v konkrétních optimalizovaných částech programu, úspora potřebné paměti a nakonec bude zhodnocen také celkový vliv na přesnost.

Na závěr se budu v kapitole 7 věnovat konečnému zhodnocení souhrnných výsledků a praktické použitelnosti práce.

Kapitola 2

Výpočty se sníženou přesností

V této kapitole se budu věnovat popisu nativních typů pro výpočty ve snížené přesnosti. Přitom budou také řešena jejich omezení a srovnání s běžnými datovými typy s jednoduchou a dvojnásobnou přesností. Dále se pak budu věnovat potenciálu zrychlení při použití poloviční přesnosti výpočtů a seznámení se základy správné práce pro optimalizaci výpočtu.

2.1 Datové typy

Kromě běžných datových typů `float` s jednoduchou přesností a `double` s dvojitou přesností lze při výpočtech na GPU pracovat s několika dalšími typy se sníženou přesností, jako je `half`, `brain float`, nebo novější osmibitové typy. Jejich reprezentace je také graficky znázorněna obrázkem 2.1 níže.

2.1.1 Float

Jedná se o typ reprezentující desetinné číslo s tak zvanou jednoduchou přesností. Je nejběžněji používaným datovým typem na grafických kartách, jelikož nabízí velký rozsah i slušnou přesnost. Ta je dostatečná pro takřka veškeré běžné typy výpočtu, proto je architektura moderních grafických karet ve velké míře postavena kolem tohoto typu ve snaze o optimální výkon při jeho použití. To se projevuje například na velikosti načítaných dat při jedné instrukci z globální paměti, či velikosti bank ve sdílené paměti.

Je definován standardem `IEEE 754` pod jménem `binary32` jakožto 32bitový typ. Začíná jedním znaménkovým bitem, pokračuje osmi bity exponentu a končí 23 bity mantisy. Exponent zde má bias -127, díky čemuž může nabývat hodnot v rozsahu od -127 do 127, přičemž hodnota 128 je rezervovaná pro speciální hodnoty. Mantisa dále začíná jedním nepsaným implicitním jedničkovým bitem pro normalizovaná čísla a nulovým bitem pro denormalizovaná čísla. Denormalizovaná čísla jsou specifická hodnotou exponentu -127. Implicitní nulový bit v praxi pro subnormální hodnoty bohužel snižuje přesnost o 1 bit. [4]

Díky výše uvedené implementaci lze pomocí tohoto typu vyjádřit maximální hodnotu ∼3,39e+38, minimální kladnou normalizovanou hodnotou je pak ∼1,18e-38 a minimální kladnou subnormální hodnotou je ∼1,40e-45. Pokud bychom chtěli spočítat sumu následující harmonické řady $\sum_{n=1}^{\infty} 1/n$, která v oboru reálných čísel diverguje k nekonečnu, v tomto formátu dojdeme po 2,097,152 iteracích ke konečné sumě v hodnotě pouhých ∼15,404. K dalšímu zvyšování sumy nedojde kvůli příliš velkému rozdílu v exponentu sumy a hodnoty dalšího prvku, což vede na praktické zaokrouhlení prvku při sčítání na hodnotu 0. [19]

2.1.2 Half

Jak již název napovídá, tento datový typ má poloviční přesnost ve srovnání s typem `float`. Motivací pro jeho vznik bylo nabídnutí možnosti urychlení výpočtů, které jsou relativně nenáročné na rozsahy hodnot a jejich přesnost. Podporu mají již od verze CUDA 3.0 a grafických karet Nvidia generace Fermi [26]. Zvýšené popularity a využití se mu však dostalo v posledních letech díky rozvoji v oblastech neuronových sítí a hlubokého učení.

Obdobně jako u datového typu `float` je tento typ definován standardem `IEEE 754` pod jménem `binary16` jakožto 16bitový typ. V tomto případě má exponent pouze 5 bitů a jeho bias je -15. Může tak nabývat hodnot pouze z rozsahu -15 až 15, což signifikantně snižuje rozsah hodnot, které jím lze vyjádřit. Pro maximální možnou hodnotu 16 i v tomto případě platí, že je rezervovaná pro speciální hodnoty. Mantisa se zde skládá z 10 explicitních bitů a také zde platí, že začíná navíc implicitní jedničkou pro normalizovaná čísla a implicitní nulou pro denormalizovaná čísla, která mají minimální možný exponent s hodnotou -15. [4]

Tato implementace umožňuje ve srovnání s vysokými čísly formátu `float` vyjádřit maximální hodnotu pouze ve výši 65504. Minimální kladná normální hodnota je zde ∼6,10e-05 a minimální kladná subnormální hodnota je ∼5,96e-08. Ze snížené přesnosti zde vyplývá, že při počítání stejné harmonické řady čísel jako výše $\sum_{n=1}^{\infty} 1/n$ zde dosáhneme pouze hodnoty ∼7,0859, a to již po pouhých 513 krocích výpočtu. Počet kroků tak je v tomto případě o více než 3 řády nižší, než byl potřebný počet kroků pro identický výpočet využívající typ `float`. Lze tedy vidět, že nejen rozsah, ale i přesnost je velmi limitována. [19]

2.1.3 Brain float

Tento typ byl narozdíl od předcházejících vyvinut výzkumným týmem `Google Brain`, který se věnuje hlubokému učení umělé inteligence. Vytvořen byl jako alternativa k již výše zmíněnému datovému typu `half` a jeho hlavní využití by mělo být v neuronových sítích, u kterých jsou zapotřebí vyšší rozsahy hodnot, ale není nutné, aby byla jejich přesnost příliš vysoká. Ač byl definován již v roce 2011, nativní podporu v grafických kartách od společnosti Nvidia dostal až od generace Ampere, která byla uvedena teprve v roce 2020. Dřívější grafické karty tak mají pouze simulovanou softwarovou podporu. [14]

Datový typ `brain float` má stejně jako typ `half` 16 bitů. Narozdíl od něj však má 8bitový exponent, tedy stejný, s jakým se setkáváme u datového typu `float`. Díky tomu jím lze vyjádřit obdobný rozsah hodnot. Hlavní nevýhodou je zde mantisa, která má pouze 7 bitů. Dochází tak k ještě větší ztrátě přesnosti ve srovnání s typem `half`. S trochou nadsázky lze prohlásit, že je tento typ ve své podstatě stejný jako `float`, ale s oříznutou mantisou. Motivaci za touto podobností lze hledat ve snaze hardwarové optimalizace. Jak uvádí [2], kratší mantisa vede například ke snížení nutné fyzické velikosti křemíku pro výpočty s tímto typem. Ve srovnání s výše uvedenými typy je zde však jeden poměrně důležitý rozdíl v tom, že tento typ nemá podporu pro denormalizovaná čísla.

Jak již bylo zmíněno, v této implementaci lze vyjádřit podobný rozsah hodnot, jako je možné při využití datového typu `float`. Konkrétní maximální hodnota je zde ∼3,39e+38 a minimální kladná hodnota je zde ∼1,18e-38. Nelze zde nicméně vyjádřit menší kladné hodnoty pomocí denormalizovaných čísel. Vzhledem k ještě více zkrácené mantise se výpočet sumy harmonické řady $\sum_{n=1}^{\infty} 1/n$ v tomto případě ustálí již po pouhých 65 krocích na hodnotě ∼5,0625. Počet kroků je tedy ještě o řád nižší, než je potřebný počet kroků u výpočtu s využitím datového typu `half`. [19]

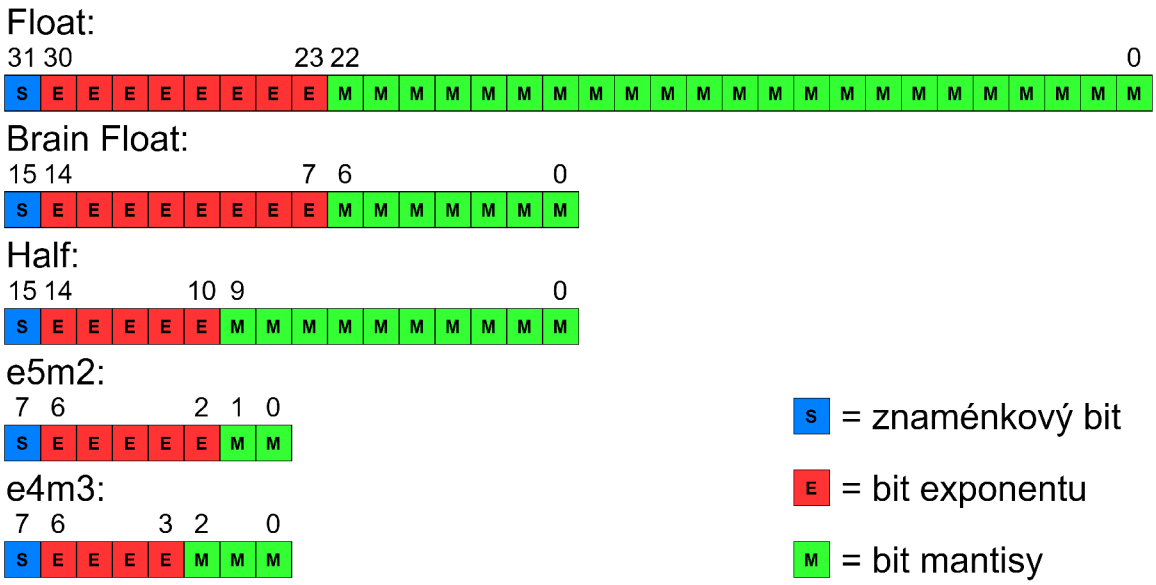
2.1.4 8bitové floaty

Nvidia, Arm a Intel ve spolupráci publikovali také osmibitové specifikace pro standardizaci [25]. Definované formáty jsou nyní konkrétně ve dvou různých kódováních a měly by sloužit dalším optimalizacím a redukci využití paměti s primárním využitím v hlubokém učení. Design těchto typů by se měl také konzistentně držet konvencí standardu `IEEE 754`. [5]

První ze dvou typů je zde navržen s účelem, aby měl obdobný numerický rozsah, jako má datový typ `half`. Proto má stejně dlouhý exponent, tedy v délce 5 bitů. Je tedy prakticky jeho zkrácenou, či oříznutou verzí. Nicméně to znamená, že mantisa zde má pouhé 2 bity, což je ve srovnání s typem `half` o celých 8 bitů méně, přičemž už pouze tříbitová redukce ve formátu `brain float` se projevila signifikantním snížením přesnosti, jak je vidět na výše uvedených příkladech se sumou harmonické řady. Maximální hodnota tohoto typu je 57344, minimální kladná normální hodnota je 2e-14 a minimální subnormální hodnota je 2e-14. [5]

Druhý z definovaných formátů nabízí exponent o délce 4 bity a mantisu o délce 3 bity. Nabídne tedy o něco vyšší přesnost, ale rozsah je ještě řádově zkrácen. Navíc má tento typ využívat většinu hodnot, které ostatní typy využívají pro speciální hodnoty. Pro hodnotu "nečíslo"(angl. not a number) je tak využita pouze 1 hodnota, hodnota nekonečno zde není dokonce žádným způsobem zastoupena. I přes tyto optimalizace však odpovídá maximální hodnota pouze číslu 448, minimální kladná normální hodnota vyjadřuje 2e-6 a minimální subnormální hodnota reprezentuje 2e-9. [5]

Tyto 8bitové typy jsou v současnosti implementovány, vzhledem k jejich nedávnému původu, pouze na grafických kartách generace Hopper společnosti Nvidia. Z toho důvodu nebudou moct být využity v rozsahu této práce. V závěru se nicméně pokusím zhodnotit, zda by mohlo být vhodné je v budoucnu využít pro další optimalizace a případně ve kterých využívaných fyzikálních charakteristickách.



Obrázek 2.1: Porovnání datových typů

2.2 Potenciál zrychlení a úvod do optimalizace

Na celkové zrychlení mají primární vliv 2 odlišné principy. Prvním je datová propustnost. Jelikož mají data pouze poloviční velikost, mělo by i jejich načtení pro výpočty trvat pouze přibližně poloviční dobu. To může hrát velkou roli hlavně v aplikacích, které jsou náročné na datovou propustnost a výpočtů samotných nad jednou načtenými daty není příliš vysoké množství. Druhým principem, který má vliv na časovou náročnost je provádění samotných instrukcí. Jak se lze dovědět z publikací společnosti Nvidia k jejich novějším grafickým kartám, generace grafických karet Pascal [12], Turing [13], Ampere [14] a Hopper [17] mají teoretický dvojnásobný výpočetní výkon při využívání poloviční přesnosti ve srovnání s jednoduchou přesností. Za jednotku času tedy lze provést dvojnásobné množství výpočtů v poloviční přesnosti. Ačkoliv je datový typ `half` podporovaný již od generace Fermi, pro starší generace, než je generace Pascal, by tento aspekt neměl ovšem hrát žádnou roli. Obdobně by teoreticky neměl dosahovat výrazného zrychlení na základě tohoto aspektu typ `brain float` u starších generací, než je generace Ampere. Tento aspekt je rovněž ukázán v následující kapitole 3 na testu násobení matic, který byl proveden na kartách Nvidia A100 generace Ampere a Nvidia V100 generace Volta. Výsledky tohoto testu jsou prezentovány grafy 3.11 a 3.12. Dále může mít na zrychlení teoreticky dopad samotné použití menších typů vzhledem k nižší náročnosti na hardwarové prostředky z hlediska využití registrů, což může v některých případech umožnit spouštění více bloků najednou, nebo omezit využití poměrně pomalé globální paměti.

K docílení optimálního výkonu nestačí ovšem pouze použít menší datové typy. Jak se lze detailně dočíst například ve článku [20], samotné použití typu `half` přináší oproti typu `float` pouze marginální zrychlení. Důvodem je, že spuštění 16bitové verze instrukce a 32bitové verze instrukce má prakticky totožnou latenci. Pro docílení plnohodnotného zrychlení je zapotřebí využívat strukturu dvou 16bitových typů, v rámci jedné instrukce lze následně provádět vektorovou operaci nad páry hodnot najednou. Pro správné čtení a zápis je navíc nutné, aby byla data zarovnána na velikost 32 bitů. V opačném případě by musela být načtena a uložena každá hodnota zvlášť pomocí dvou instrukcí. CUDA API pro tyto potřeby v současné době poskytuje nativní datové typy `half2` a `__nv_bfloat162`.

CUDA API je mimo jiné optimalizované pro práci s datovými typy `float2` a `float4`. Pro jejich ideální využití nabízí 64 a 128bitové instrukce pro načtení dat a zápis do paměti [24]. Různé zdroje se však liší v názorech na to, zda je vhodné používat pro jemné optimalizace tyto typy, nebo zda se držet obecného doporučení používat strukturu polí jednoduchých typů s délkou slova 32 bitů a který přístup by měl přinést ideálnější výkon. Transakce pro celý warp vláken by přitom měly mít velikost 32, 64, nebo 128 bytů [16]. Při dnes typické velikosti warpu 32 vláken a použití 128bitových instrukcí je tak zapotřebí 512 bytů dat, což při zarovnaném přístupu vede přinejmenším na 4 transakce o velikosti 128 bytů. Z toho důvodu budou při testech v následující kapitole využity také tyto struktury a bude ekvivalentně testována potenciální možnost použití vlastních obdobných struktur zapouzdřujících čtveřice a osmice 16bitových typů.

Kapitola 3

Testování výkonu v jednoduchých aplikacích

Tato kapitola bude věnována performačním testům několika jednoduchých aplikací, které by měly pomoct s navržením vhodných způsobů optimalizace v reálném produktu. Z testů by měl být vidět vliv paměťové propustnosti při použití různých typů, ale také časová náročnost výpočetních instrukcí při využití různých datových typů. Všechny testy byly prováděny na grafických kartách Nvidia V100 generace Volta a Nvidia A100 generace Ampere. Pro všechny testy byly také použity stejné parametry spuštění se 320 bloky o velikosti 320 vláken na blok.

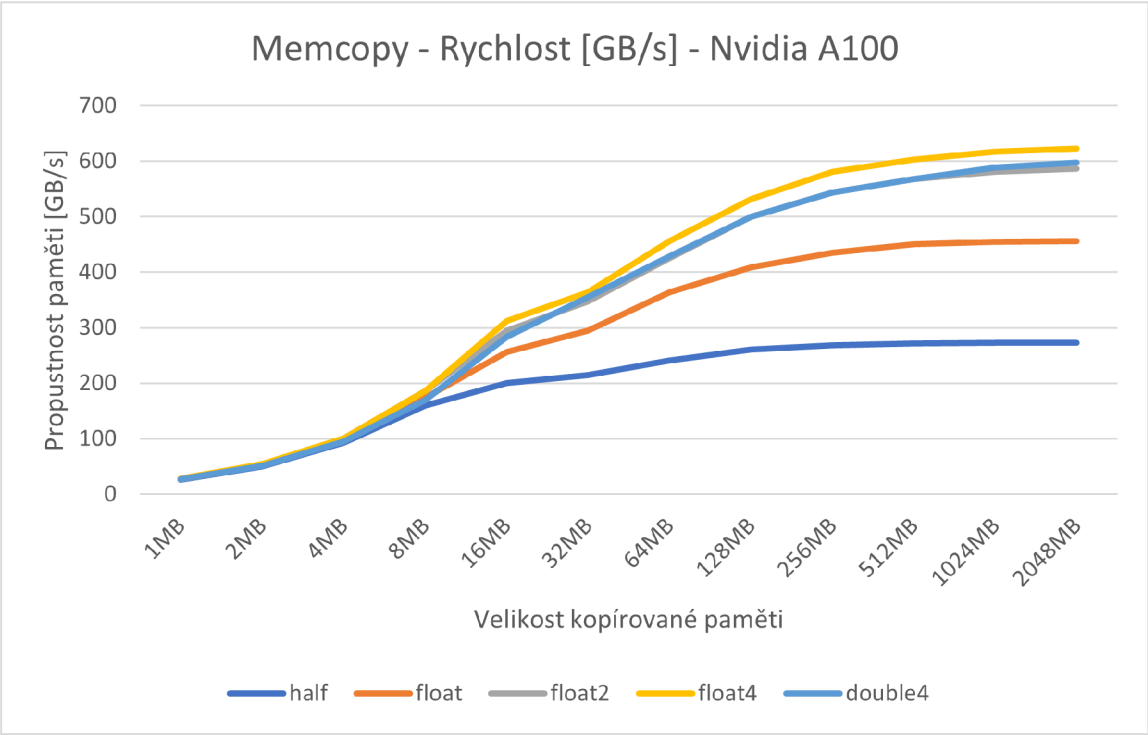
3.1 Kopírování dat

Tento test provádí jednoduché kopírování dat z jednoho pole do druhého. Pole jsou alokována postupně v různě velkých datových typech od velikosti 2 bajty pro typ `half` do velikosti 32 bytů pro typ `double4`. Účelem testu bylo zjistit datovou propustnost při použití datových typů různých velikostí. Zároveň byl použit k určení vhodného počtu spouštěných bloků a množství vláken na blok pro zajištění reprezentativních výsledků.

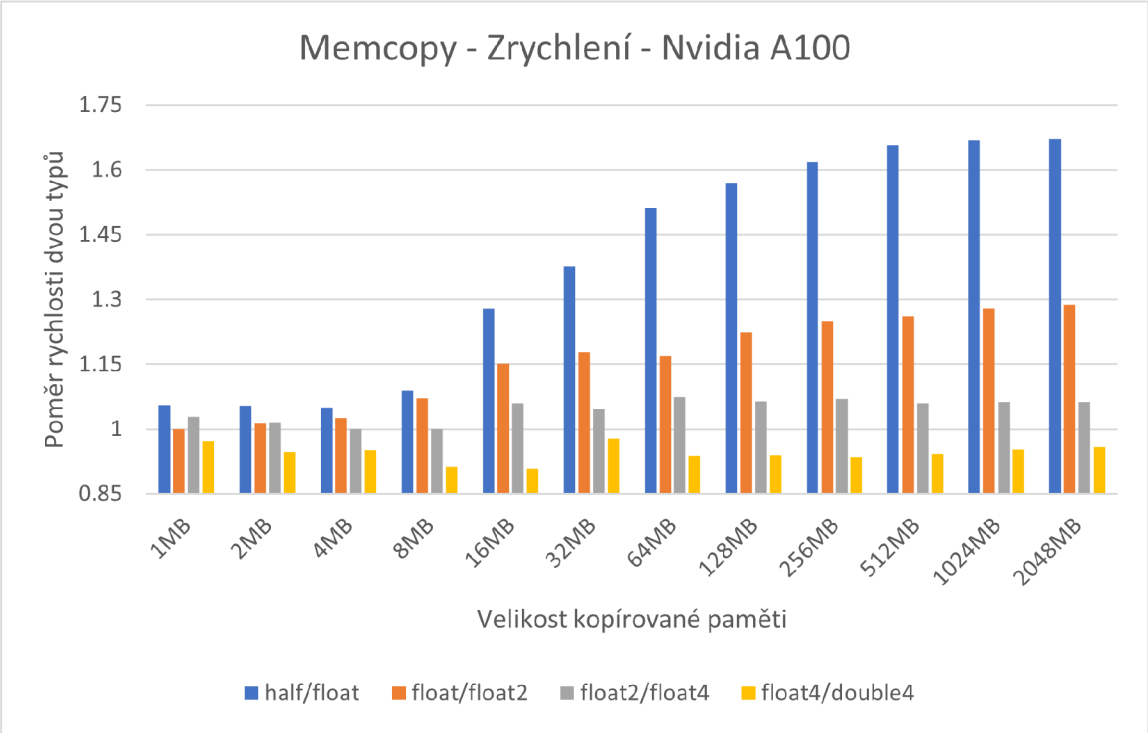
S účelem rozumného měření času takto jednoduchých programů a přitom škálování odpovídajícího očekávání byly vyhodnoceny jako rozumné parametry 320 bloků o 320 vláknech pro GPU Nvidia A100. Při menším množství spuštěných vláken nebyla dostatečně využita propustnost paměti. V důsledku tedy mohlo vypadat používání 32 či dokonce 64 bytových typů, které vyžadují několik instrukcí na vlákno a transakcí na warp vláken, jako výrazně efektivnější varianta.

Pro GPU Nvidia V100 by stačilo používat pro podobné výsledky, jako jsou prezentovány u grafické karty Nvidia A100 pouze 128 bloků o velikosti 128 vláken. Budou nicméně použity stejné velikosti parametrů jako u výše zmíněné karty Nvidia A100, tedy 320 bloků s velikostí 320 vláken. Bude tak lépe vidět škálování aplikací i ve druhém případě, kdy rozdíly v propustnosti přestávají být tak vysoké.

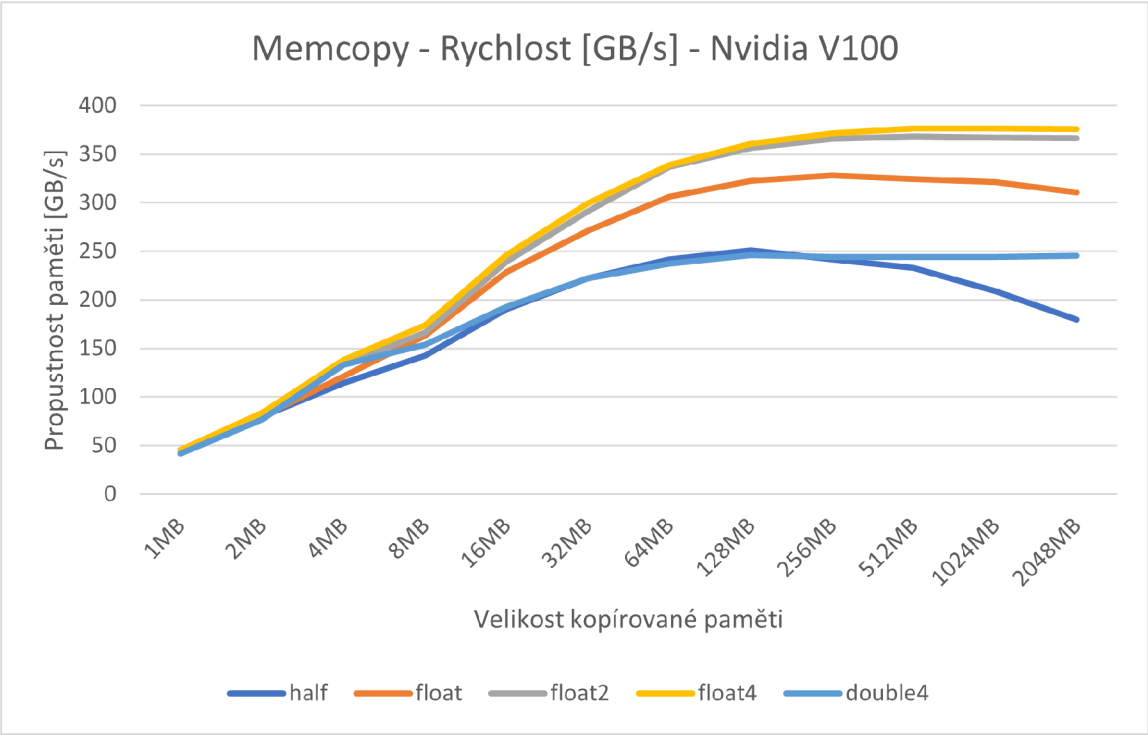
Jak lze vidět na grafech 3.1 a 3.3, pro malá data do velikosti 8 MiB u GPU Nvidia A100 a 4MiB u Nvidia V100 příliš nezávisí na velikosti použitého datového typu. To lze vysvětlit tím, že u malého množství dat je paměťová propustnost a paralelismus grafických karet natolik vysoký, že dojde k splnění požadavku s takřka totožnou latencí nehledě na konkrétní velikost a počet transakcí. To je ještě více umocněno tím, že se veškerá data vejdou do rychlé paměti cache. Rozdíly se však začnou projevovat až při vyšších velikostech dat.



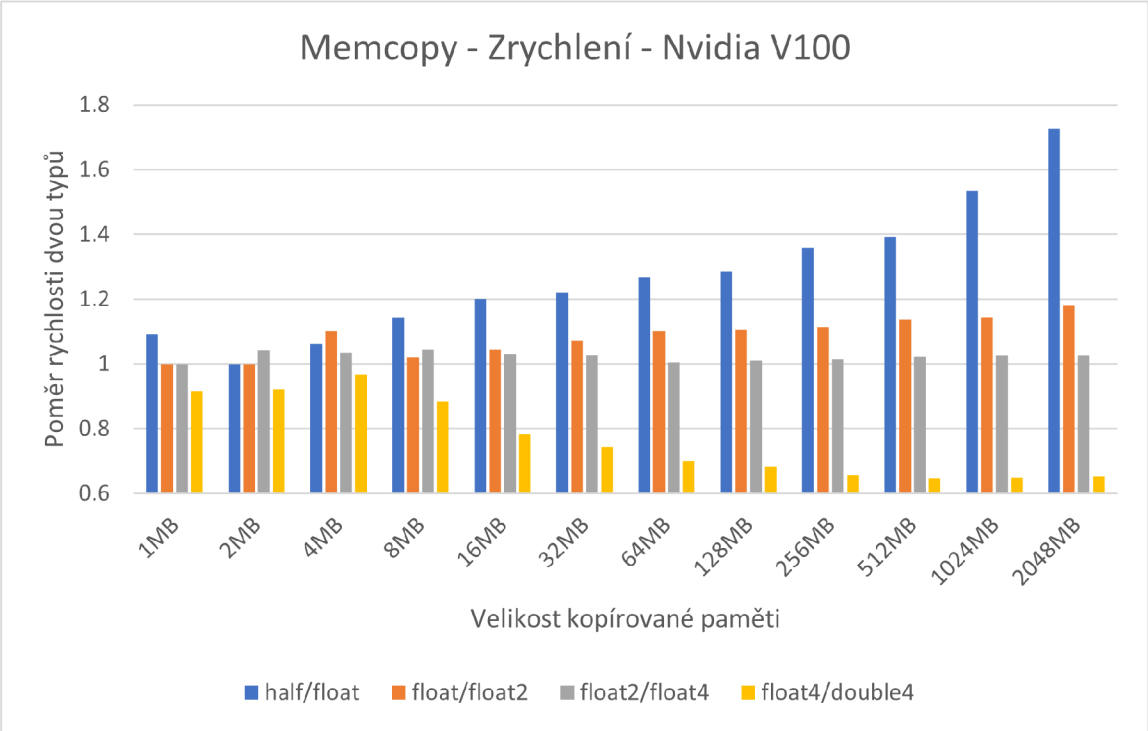
Obrázek 3.1: Graf rychlosti kopírování dat dle velikosti datového typu - Nvidia A100



Obrázek 3.2: Graf zrychlení kopírování dat dle velikosti datového typu - Nvidia A100



Obrázek 3.3: Graf rychlosti kopírování dat dle velikosti datového typu - Nvidia V100



Obrázek 3.4: Graf zrychlení kopírování dat dle velikosti datového typu - Nvidia V100

Lze vypozorovat, že pro dobrou datovou propustnost určitě nelze používat 2 byty velký typ `half`. Ten v případě velkého množství použité paměti na starší architektuře Volta dokonce trpí na postupné snižování propustnosti, viz graf 3.3. Tento jev zpomalování lze vidět dokonce i na 4 byty velkém typu `float`, nicméně v takovém případě není zdaleka tak zásadní. Navíc i použitím datového typu `float2` lze dosáhnout nezanedbatelně lepšího výsledku ve srovnání s typem `float`, jak lze vidět hlavně na grafu 3.2. Použití datového typu `float4` vede na další malé zrychlení, které již nicméně není tak markantní.

Další testování navíc ukázalo, že se zvětšujícím se počtem spuštěných vláken dochází obecně ke zmenšování rozdílů, jelikož i menší datové typy začnou dosahovat lepších propustností. Obzvlášť rozdíl mezi typy `float2` a `float4` začne tedy být zanedbatelný. Tohoto chování si lze všimnout hlavně na grafu 3.4, jelikož jak bylo výše zmíněno, pro tuto grafickou kartu lze ideální škálování vidět již při řádově nižším počtu spuštěných vláken. Na závěr se hodí zmínit, že použití větších datových typů, jako je například nativní typ `double4` s velikostí 32 bytů, nemá praktické využití pro optimalizaci propustnosti paměti. Při spuštění kernelu s rozumně vysokým množstvím vláken je jeho použití naopak kontraproduktivní.

3.2 Redukce

V tomto testu byl naimplementován jednoduchý program, který na grafické kartě spočte sumu zadaných čísel. Ukládání výsledků je vždy do typu `float`. Konkrétně se v každém vlákně vypočte parciální suma, následuje sečtení parciálních sum v rámci warpů a výpočet je ukončen atomickým přičítáním těchto parciálních sum z každého warpu do výsledku. Při menších vstupech s velikosti řádově několik megabajtů docházelo k relativně velkým rozptylům časů mezi jednotlivými spuštěními. Pro zajištění vyšší konzistence tedy byl kernel, který má redukci spočítat, proveden stokrát v rámci jednoho spuštění programu. To mohlo v některých ohledech ovlivnit výsledky především u nižších velikostí kvůli využití paměti L2 cache. Tento aspekt bude proto nutné zohlednit ve zhodnocení výsledků testu.

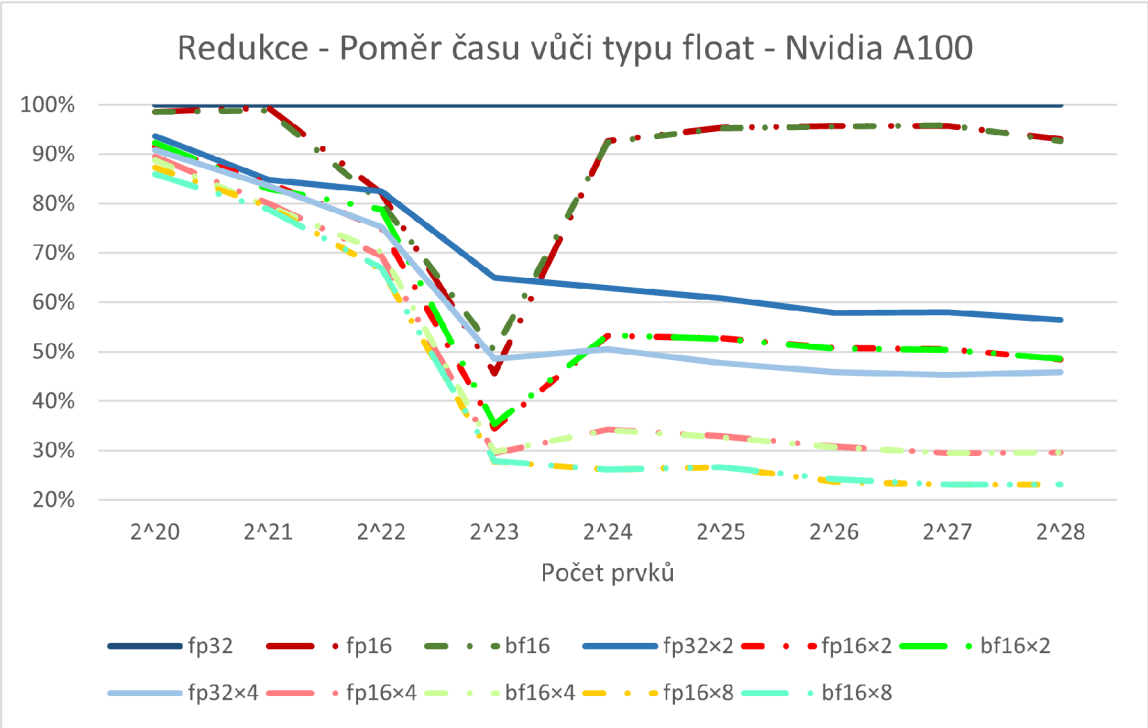
Jak již ukázaly minulé výsledky při testování věnovaného kopírování dat, pro co nejideálnější možnou datovou propustnost je vhodné používat větší datové typy, jako je například `float2` a `float4` s velikostí 8 a 16 bytů. Jelikož nativní datové typy pro poloviční přesnost obsahují maximálně dvojice hodnot a mají proto tedy maximálně 4 byty, byly pro testování naimplementovány struktury obsahující dvojici a čtveřici těchto struktur dvojic s poloviční přesností. Vlákno v případě použití těchto struktur nejprve sečte n-tici prvků a teprve tento výsledek následně přičítá k parciální sumě.

Jak lze vyčíst z grafu 3.5, při menším datasetu (do osmi milionu prvků) ovlivňuje silně L2 cache výsledky tohoto testu. Datová propustnost je díky ní natolik vysoká, že se do jisté míry ztrácí rozdíly v použití některých typů. Pro 8 milionů (tj. 2^{23}) prvků zde ovšem nastává zajímavá situace, kdy pouhé použití typu `half`, či `brain float` vede na zkrácení času výpočtu na méně než polovinu. Důvodem patrně je, že použití většího typu v důsledku znamenalo, že se již data nevlezla do rychlé paměti cache. Pouhé použití menších datových typů tedy může v některých případech razantně zlepšit i samotnou práci s pamětí cache. Dále je zajímavé, že použití typů s poloviční přesností mělo méně, než desetiprocentní vliv na výkon pro vyšší objemy dat. Ač mají instrukce teoreticky nižší latenci a bylo nutné použít jen poloviční velikost celkových dat, je zde poznat velmi špatná datová propustnost. Ve srovnání se základními typy s poloviční přesností vypadá situace mnohem pozitivněji například při použití dvojic a čtveřic datového typu `float`. Pro dvojice bylo dosáhnuto zrychlení až ∼1,77, což je v grafu vyobrazeno jako zkrácení času na ∼56,5 %. Pro čtveřici se čas stabilizoval dokonce kolem 46 % ve srovnání s typem `float`. To jsou dokonce lepší vý-

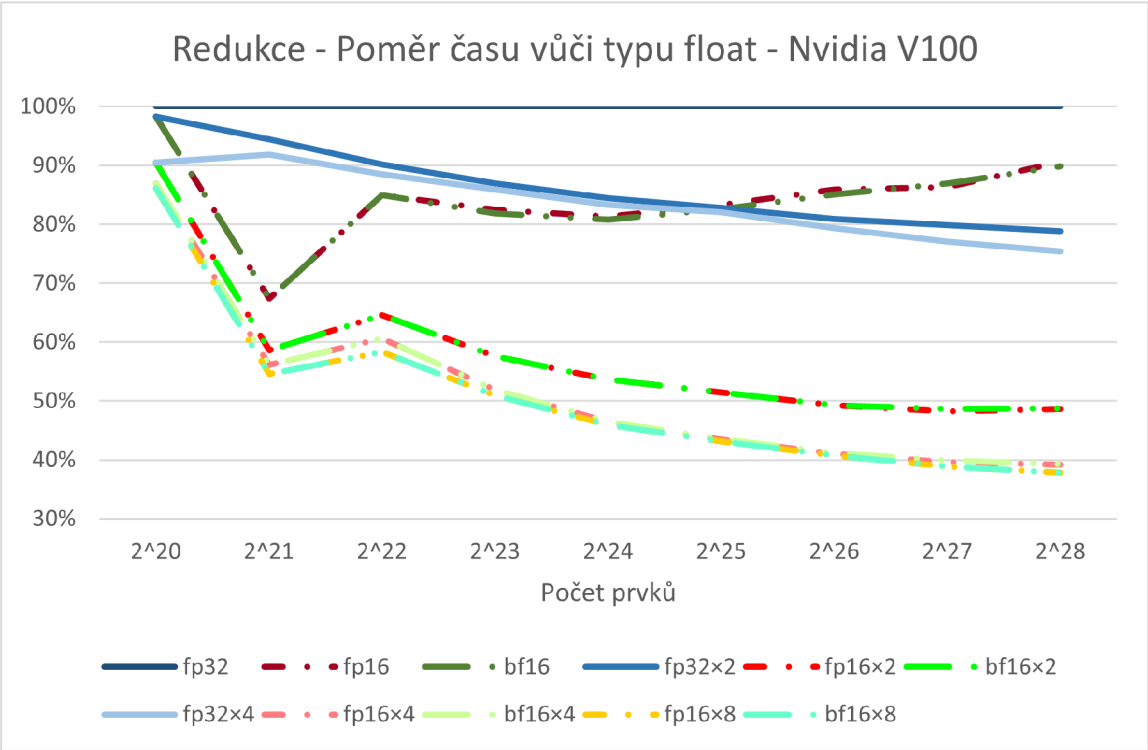
sledky, než jaké byly očekávány na základě výše uvedeného grafu 3.1, kde čtveřice dosahuje pouze o zhruba třetinu vyšší propustnosti paměti. Ještě lepších výsledků však bylo docíleno za pomoci dvojic, čtveřic a osmic typů s poloviční přesností. Ač jsou tyto dvojice o trochu pomalejší ve srovnání s výkony, kterých bylo dosáhnuto se čtveřicí typu `float`, jsou velmi blízko a zároveň je kód méně komplikovaný. V důsledku může být použití dvojic také snazší v reálném kódu. Navíc tato dvojice spotřebovává pouze čtvrtinovou paměť registrů, nebo také stejnou jako základní implementace s typem `float`. Pokud by byly použity čtveřice typů s poloviční přesností, dojde ke stabilizování času okolo 30 % z původního času, což je více než trojnásobné zrychlení. V případě použití osmic je zrychlení dokonce více než čtyřnásobné. To lze jinak vyjádřit také jako dvojnásobné zrychlení ve srovnání se čtveřicí `floatů`, což je očekávané zrychlení vzhledem k polovičnímu objemu dat.

Obdobně jako u grafu 3.5 lze vidět i v případě grafu 3.6 silné ovlivnění pamětí L2 cache pro nižší počty prvků. To lze pozorovat specificky pro nejnižší počet prvků v grafu a následně také u 2. nejmenšího počtu prvků při bězích výpočtů vyžívajících typy s poloviční velikostí. Opět tak platí, že pro nejmenší počet prvků jsou rozdíly jen velmi malé. Ve druhém sloupci pak platí, že dochází k velkému zrychlení na základě využití paměti cache u 2 byty velkých typů ve srovnání s typem `float`. Od 4 milionů prvků (tj. 2^{22}) pak lze vidět relativně pěkně škálující výsledky. Pro dvojice typu `float` zde dochází ke zrychlení, které má takřka lineární závislost na velikosti vstupu. To samé platí i pro čtveřice, které dosahují mírně lepších výsledků. Tento trend koresponduje s očekáváními na základě dříve dosažených výsledků obsažených v grafu 3.3, ačkoliv podobně jako u testu provedeného na grafické kartě Nvidia A100 jsou výsledky o něco lepší, než jaká byla očekávání. Ani pro nejvyšší velikost testovanou při kopírování dat totiž zvýšení paměťové propustnosti nepřesáhlo 20 %. V tomto testu se však setkáváme se zrychlením na méně než 80 % času již pro velikosti vstupu odpovídající datům o velikosti 1 GiB, což znamená, že bylo dosáhnuto více než o 25 % vyšší propustnosti. V případě základních typů s poloviční přesností bylo dosaženo lepšího zrychlení, než u druhé testované karty, avšak se zvětšujícími daty se výsledky zhoršují, což ovšem koreluje s údaji v grafu 3.3 z předešlého testu, kde lze pozorovat klesající propustnost paměti. V případě dvojic typů `half` a `brain float` výpočet dosahuje až dvojnásobného zrychlení, což je velmi podobný výsledek jako u druhé testované karty. Při použití čtveřic již zde nicméně nedochází k tak velkým zrychlením a rozdíl mezi čtveřicemi a osmicemi je zanedbatelný. Maximální zrychlení je v tomto případě pouze ∼2,5, což je nicméně stále o něco vyšší zrychlení, než jaké bylo očekáváno na základě naměřených rozdílů v propustnosti paměti vyobrazených v grafu 3.4.

Výsledky tohoto testu ukázaly, že tento případ byl silně omezen propustností paměti. Mezi typy `half` a `brain float` však byly nulové rozdíly na hraně chyby měření, a to i přestože starší z testovaných grafických karet nepodporuje typ `brain float` na hardwarové úrovni. Z toho vyplývá, že se pro ně v praxi používají 32bitové výpočty, které však byly ve srovnání s omezením propustnosti paměti zanedbatelné. V dalších testech, kde se mezi těmito typy nebudou vyskytovat rozdíly, proto budu vynechávat typ `brain float` z výsledků. Uváděn nicméně bude, pokud to bude relevantní z hlediska měřitelných rozdílů. Dále taky budou vynechány menší vstupy, které by neměly další přínosnou vypovídající hodnotu. Díky tomu by mělo být snazší dosáhnout konzistentních výsledků i bez násobného opakování výpočtu v rámci jednoho spuštění programu. Obecně byly výsledky zrychlení o něco málo lepší, než bylo očekáváné podle testů propustnosti pro různé typy. To bylo možné pozorovat například pro dvojice typu `float`. Tyto výsledky lze přisuzovat tomu, že narozdíl od testu výše zde neprobíhal okamžitý zápis do globální paměti při každém čtení, což mohlo uvolnit část přenosového pásma a následně umožnit lepší škálování pro objemnější typy.



Obrázek 3.5: Graf Zrychlení výpočtu redukce - Nvidia A100



Obrázek 3.6: Graf Zrychlení výpočtu redukce - Nvidia V100

3.3 Skalární součin

Pro tento test byla naimplementována jednoduchá suma součinů dvou vektorů. Každé vlákno si tak násobí svou část součinů a počítá z nich parciální sumu. Na konci výpočtu dojde k sumě parciálních sum.

Pro propustnost a škálování je tak oproti předchozímu testu největší rozdíl v tom, že se přistupuje ke dvěma odlišným vektorům hodnot. Jelikož je vždy potřeba u obou vektorů přistupovat na stejné indexy, tak tento typ výpočtu umožňuje také zamyšlení se nad ideálním způsobem uložení obou vektorů. Jednou variantou jsou dvě oddělená pole. Druhou potenciálně optimalizovanější variantou by však mohlo být prokládané ukládání do jednoho pole dvojic. Taková varianta bude v grafech označená sufixem `o`, `fp32o` pak například znamená uložení obou vektorů do jednoho pole dvojic typu `float`, `fp16×2o` zase označuje prokládané uložení obou vektorů do jednoho pole po dvou dvojicích typu `half`.

Při pohledu na graf 3.7 níže, který prezentuje dosažené výsledky na grafické kartě Nvidia A100, si lze všimnout několika zajímavostí. Snaha o optimalizaci uložením obou vektorů v jednom poli se zdá být efektivnější, ale pouze v některých případech.

Při použití pole dvojic typu `float` obsahující oba dva vektory tak lze vidět zkrácení potřebného času pohybující se okolo pěti procent. To je poměrně dobrý výsledek s ohledem na skutečnost, že z pohledu paměti i přesnosti výpočtů nedochází k žádným změnám. Lze tedy říct, že tento přístup zde má pouze pozitiva a jedná se o zrychlení zadarmo. V případě použití dvou polí dvojic, nebo jednoho se čtveřicemi je dosahováno zrychlení na zhruba 75 % času. Nelze již však konzistentně říct, která varianta je efektivnější.

Použití dvou polí čtveřic typu `float` pak má již pouze minimální efekt na zrychlení oproti předešlým dvěma přístupům. Samotné uložení do typu `half` dosahuje v tomto testu o cca 5 % kratších časů, než bylo dosaženo optimalizací uložením do jednoho pole typu `float`. Optimalizace do jednoho pole dvojic přináší úspěch i v tomto případě a dochází k dalšímu zkrácení času o zhruba 5 až 7 %. Celkový čas se tak pohybuje mezi 80 a 85 % ve srovnání s typem `float`.

Při použití dvou polí dvojic, nebo jednoho pole čtveřic typu `half` dochází k velmi vysokému zrychlení ve srovnání se všemi dosud uvedenými přístupy, které se pohybovaly v nejlepším případě u hranice 70 % času oproti základní neoptimalizované variantě. Dvě pole dvojic se dokázaly přiblížit k hranici 50 % času a optimalizace do jednoho pole čtveřic dokonce umožnila dokonce překročit tento práh. Obecně zde lze vidět, že optimalizace prokládaným ukládáním je v tomto případě vždy výhodnější.

Dále byly testovány dvě pole čtveřic, optimalizace těchto dvou polí do jednoho prokládaného pole osmic a dvě pole osmic typů `half`. Výsledky mezi nimi se obecně vzato příliš nelišily. Snaha o optimalizovanou variantu se však v tomto případě nesetkala s úspěchem a byla z daných variant časově nejhorší, ač rozdíly byly malé. Dvě pole osmic sice dopadly nejlépe, ale zrychlení bylo již prakticky zanedbatelné. Všechny tři poslední varianty se dokázaly dostat časově až pod hranici 40 % času původní varianty, což odpovídá ∼2,5násobnému zrychlení.

Výsledky zrychlení na druhé testované kartě Nvidia V100 jsou prezentované grafem 3.8. Opět lze vidět, že už pouhá optimalizace obou vektorů do jednoho prokládaného pole přinesla malé zrychlení. Toto zrychlení je nicméně nižší než v případě druhé testované grafické karty. Použití dvou polí dvojic pak vedlo ke zhruba o 5 % kratším časům oproti této optimalizované variantě. Snaha o optimalizaci prokládáním do jednoho pole čtveřic se nesetkala s dalším zrychlením a uložení vektorů do dvou polí čtveřic přineslo pouze zanedbatelné zrychlení. Takové výsledky jsou ovšem očekávatelné na základě grafu 3.4, kde lze vidět, že

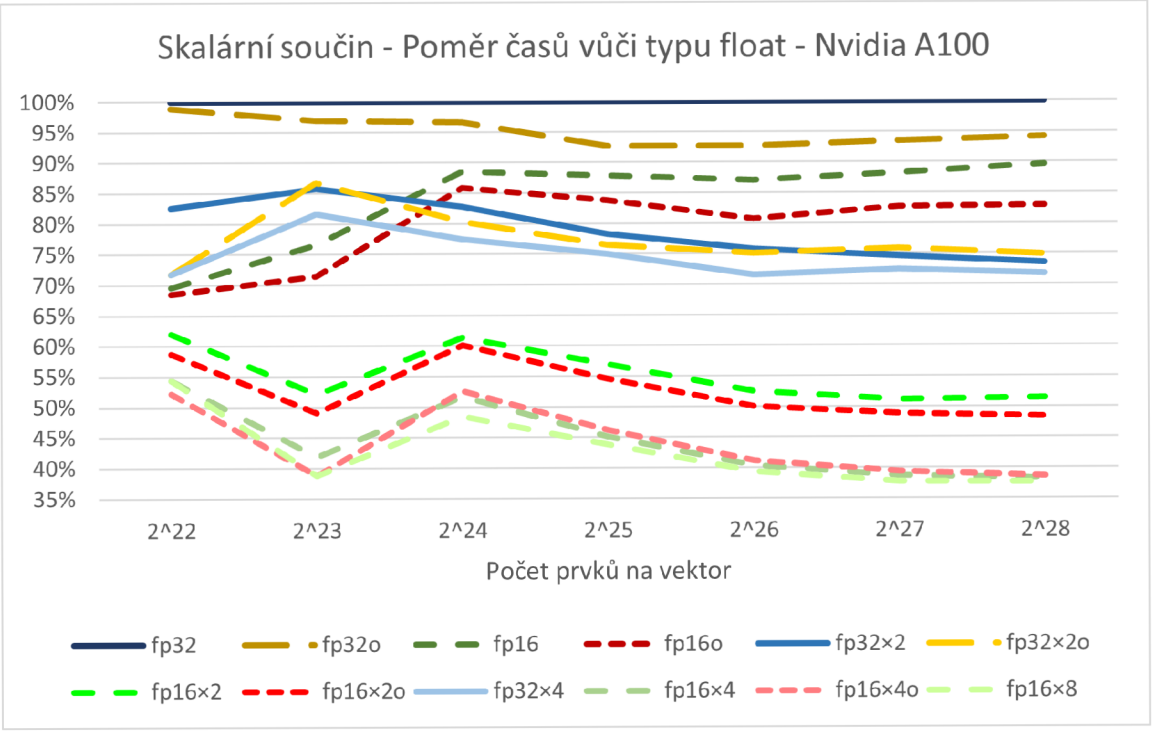
použití 16bytových struktur nepřineslo signifikantní zlepšení propustnosti paměti ve srovnání s 8 bytů velkými strukturami.

Použití typu `half` se na GPU Nvidia V100 setkalo naopak s mnohem lepšími zrychleními ve srovnání s testováním na druhé grafické kartě generace Ampere. Dle očekávání ovšem rovněž dochází pro objemná data ke zhoršování propustnosti a ke snižování zrychlení. Optimalizace prokládaným uložením obou vektorů do jednoho pole se tak zde setkává nejenom s dalším zefektivněním přístupů do paměti, ale také se signifikantním snížením tohoto efektu. I pro velké vstupy je tak potřebný čas o více než třetinu kratší ve srovnání s pouhým použitím dvou polí typu `float`. Použití dvou polí dvojic následně vede k dalšímu zrychlení a lepšímu škálování. Podobně jako v grafu 3.7 se zde blíží k hranici 50 % a optimalizovaná varianta ukládající oba vektory do jednoho pole tuto hranici dokonce překračuje pro objemnější data.

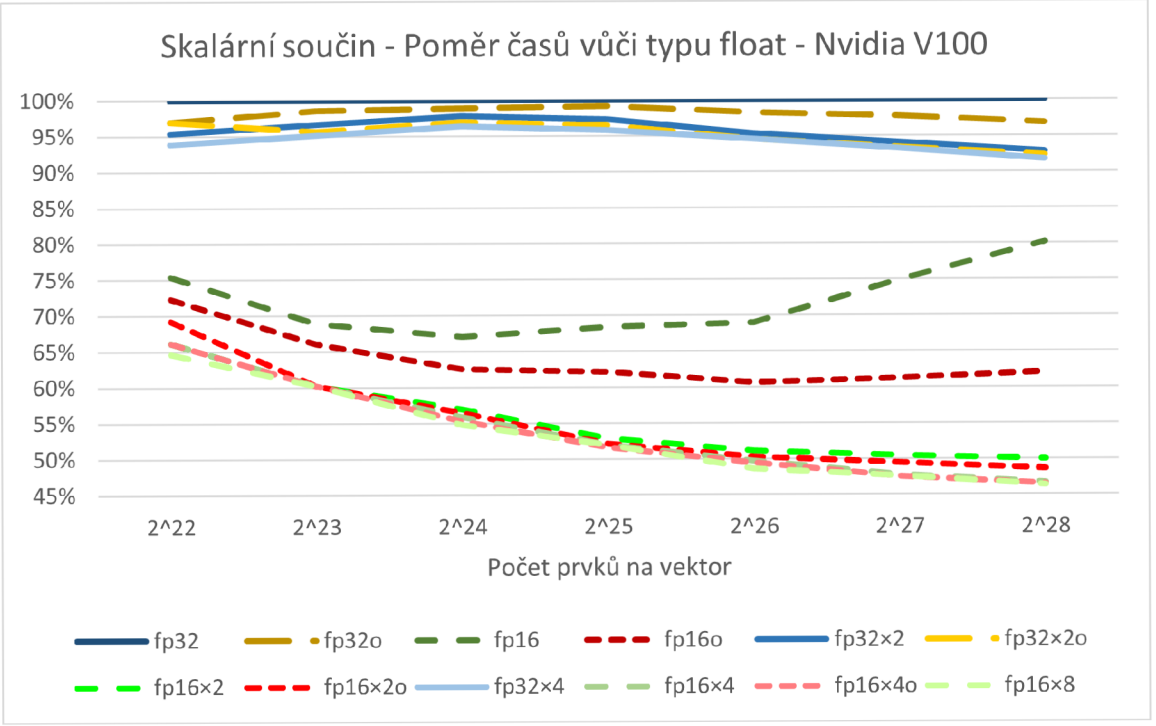
Použití čtveřic, varianta s pokusem o optimalizaci, ani použití osmic se od sebe při experimentech na testované kartě generace Volta vzájemně takřka neliší. Oproti použití dvojic navíc přináší jen zanedbatelné zrychlení. Vzhledem k ceně použití většího množství registrů a složitějšímu programování se tak nezdají tyto varianty jako výhodné a pro praxi použitelné.

Obecně tento test ukázal, že minimálně v některých případech se vyplatí uvažovat nad způsobem uložení dat v paměti. Pokud bude vždy přistupováno ke stejným indexům dvou polí, může být vhodnější je uložit jako pole struktur namísto struktury polí jednoduchých typů. Díky tomu se podařilo na obou testovaných kartách docílit zrychlení při použití typu `float` bez žádných nežádoucích efektů, byť toto zrychlení nebylo příliš signifikantní.

Jako velmi zajímavé pro optimalizace se pak ukázaly především prokládáním optimalizované verze využívající dvojice a čtveřice typu `half` pro uložení obou vektorů do jednoho pole. V případě první varianty, kde dvojice obsahuje vždy po jednom prvku z obou vektorů, bylo dosaženo především na slabší kartě Nvidia V100 velmi dobrého zrychlení, ač ani na druhém GPU Nvidia A100 nebylo zrychlení zanedbatelné. Přitom takto optimalizovaný výpočet potřebuje nižší počty registrů ve srovnání s použitím typu `float`, což může u některých aplikací pomoct k dalšímu zrychlení. Varianta ukládající oba vektory do jednoho pole po dvou dvojicích typu `half` pak dosahovala na obou kartách velmi podobného až dvojnásobného zrychlení. Takové využití přitom zachovává stejnou náročnost výpočtu na registry, jako potřebuje základní varianta využívající jednoduchou přesnost. Pokud by však byl vyžadován jiný typ přístupu, než je přístup vždy na stejné indexy pro oba vektory, tak pořád platí, že přístup po dvou dvojicích s poloviční přesností dosahuje téměř tak dobrých zrychlení. Snaha o tento typ optimalizace pro větší struktury a jednoho přístupu po 16 bytech namísto dvou přístupů po 8 bytech se naopak v tomto případě nesetkala se zlepšením. Varianty použití čtveřic s jednoduchou přesností a osmic s poloviční přesností v obou případech rovněž nepřinesly velké zlepšení.



Obrázek 3.7: Graf Zrychlení výpočtu skalárního součinu - Nvidia A100



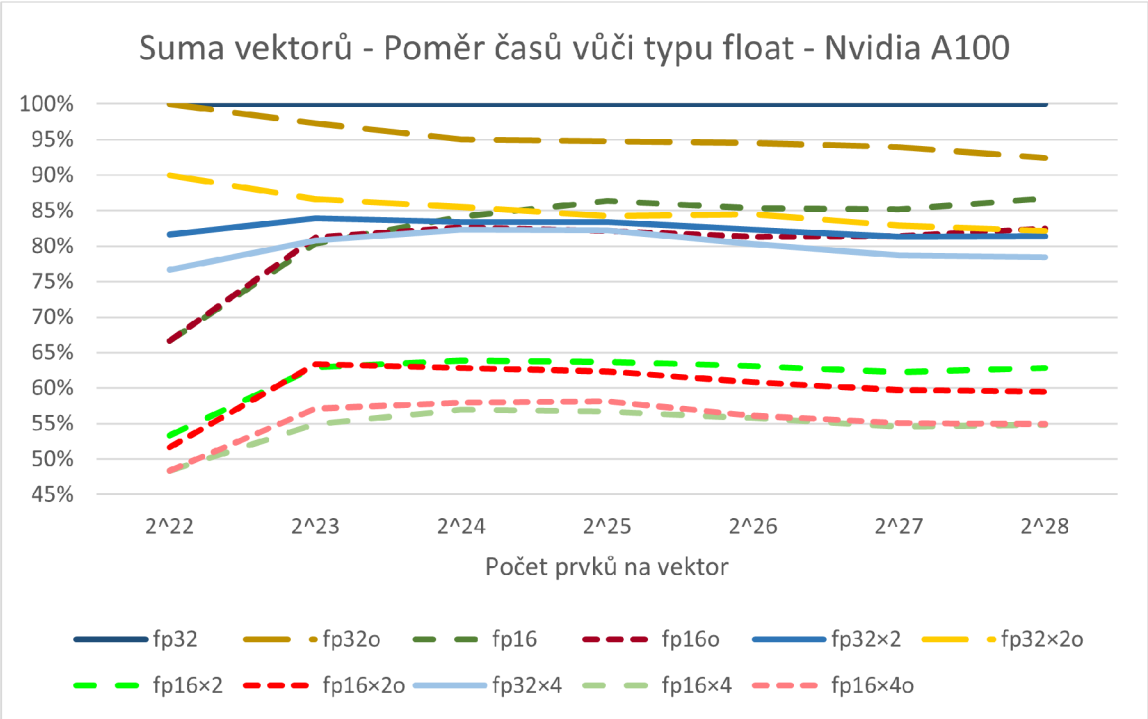
Obrázek 3.8: Graf Zrychlení výpočtu skalárního součinu - Nvidia V100

3.4 Suma vektorů

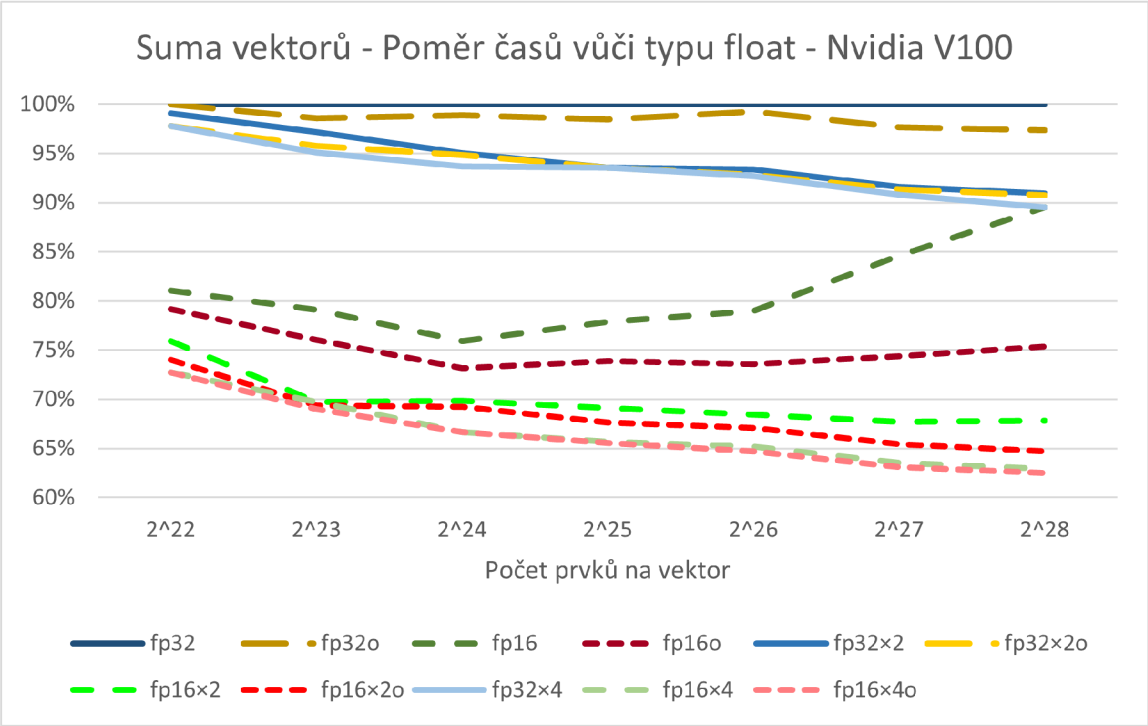
Pro potřeby tohoto testu byl napsán program, který přijímá jako vstup dva vektory a výsledek jejich součtu ukládá do třetího vektoru. Ve srovnání s předchozím testem, kde byl dělán výpočet skalárního součinu, je tak zapotřebí navíc výstupní pole. To bylo v předešlých testech zapotřebí pouze v úvodním testu kopírování paměti. Díky tomu bude část přenosového pásma globální paměti využita také pro samotné ukládání výsledků, bude tedy zajímavé srovnat výsledky s testem výpočtu skalárního součinu. Obdobně jako u předešlého testu budou i zde testovány varianty se snahou o optimalizaci prokládaným uložením obou vstupních vektorů do jednoho pole. Varianta dvou polí osmic typů s poloviční přesností zde bude nicméně oproti předešlému testu vynechána, jelikož by bylo třeba zapisovat výsledky po osmicích s jednoduchou přesností. Na základě předešlého testování lze přitom dopředu odhadnout, že by takový přístup měl negativní efekt na přenosové pásmo. To je ostatně rovněž ukázáno v grafech 3.2 a 3.4, které ukazují významné zpomalení při použití struktur `double4` ve srovnání s menšími strukturami `float4`.

Graf 3.9 ukazuje, že na grafické kartě Nvidia A100 lze opět dosáhnout zrychlení pohybujícího se okolo 5 % již pouhou optimalizací dvou vstupních vektorů do jednoho pole. Tento případ zde také dobře škáluje a obecně jsou tak výsledky pozitivní hlavně pro objemnější data. Použití dvou polí dvojic typu `float` pro vstupní pole pak vede ke zkrácení času o necelých 20 %. Snaha o optimalizaci do jednoho pole se dvěma dvojicemi `floatů` na prvek ovšem ani zde nevede ke zlepšení. Z grafu se nicméně zdá, že taková varianta lépe škáluje a pro extrémně velké vstupy by tak mohla být efektivnější. Použití dvou polí čtveřic pak přináší úsporu času o dalších ∼5 %. Dle očekávání přináší jednotlivé varianty využívající jednoduchou přesnost nižší zrychlení, než u testu skalárního součinu. Jednoduchá záměna vstupních dat za data s poloviční přesností vedle toho přinesla překvapivě dokonce vyšší úsporu času pohybující se okolo 15 %. Uložením do jednoho pole pak bylo zrychlení dále zvýšeno především pro větší vstupy. U menších vstupů je mezi těmito dvěma variantami rozdíl na hranici chyby měření. Použití dvou polí dvojic typu `half` pak přináší zhruba třetinovou úsporu času oproti základní variantě a optimalizace uložením do jednoho pole přináší další zrychlení, které se v cíli pohybuje u hranice 60 % potřebného času. Dle očekávání je tedy v tomto případě zrychlení o něco nižší ve srovnání s předešlým testem, kde lze na grafu 3.7 vidět zkrácení potřebného času na pouhých 50 %. Použití čtveřic s poloviční přesností přináší zkrácení nutného času na pouhých 55 % a opět platí, že snaha o optimalizaci pomocí uložení do jednoho pole se zde efektivně nesetkává s úspěchem z pohledu zrychlení.

Graf 3.10 znázorňuje dosažené výsledky pro GPU Nvidia V100. I zde je dosaženo drobného zrychlení uložením vstupních vektorů do jednoho prokládaného pole. Při použití polí s dvojicemi je pak dosahováno překvapivě lepších zrychlení ve srovnání se skalárním součinem, jehož výsledky prezentuje graf 3.8 výše. Uložení obou vektorů do jednoho pole po dvou dvojicích pak přineslo pouze statisticky zanedbatelné zlepšení. Obdobně ani použití čtveřic nepřineslo významné zrychlení. Pro všechny tyto varianty platí, že zrychlení poměrně stabilně stoupá s velikostí vstupu. V případě použití typu `half` lze opět vidět vyšší zrychlení hlavně pro malé vstupy. Stejně jako u předešlého testu také platí, že uložením do jednoho pole dochází k významnému zlepšení charakteristiky pro velké vstupy. Čas je v takovém případě redukován na přibližně 75 %. Uložení po dvou prvcích i tentokrát vede ke zrychlení, přičemž varianta uložení do jednoho pole vede také na trochu lepší škálování s velikostí vstupů. Použití čtveřic pak vede na další zrychlení, které však již není příliš signifikantní. Optimalizace uložením do jednoho prokládaného pole ovšem přináší narozdíl od předešlých testů rovněž nějaké zlepšení.



Obrázek 3.9: Graf Zrychlení výpočtu sumy vektorů - Nvidia A100



Obrázek 3.10: Graf Zrychlení výpočtu sumy vektorů - Nvidia V100

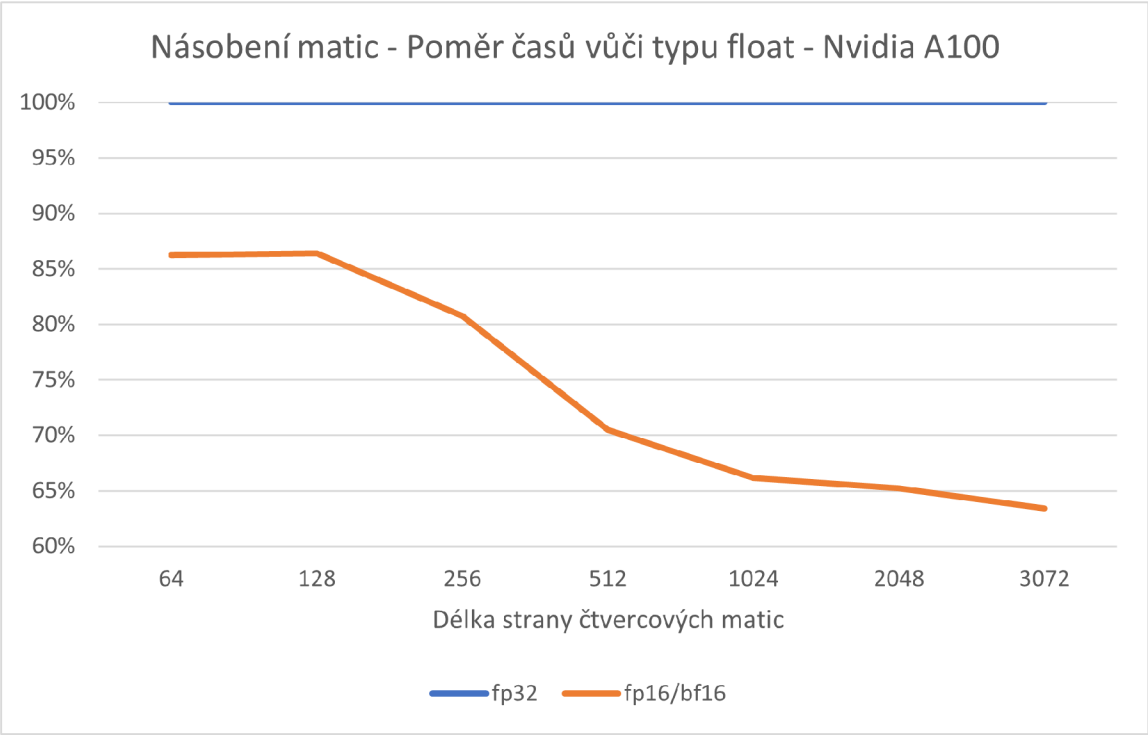
Dosažené výsledky prezentované v grafech 3.9 a 3.10 byly víceméně dle očekávání konzistentní s předešlými testy. Proti očekáváním bylo pouze v některých případech dosaženo lepších zrychlení, než bylo dosaženo v testu počítající skalární součin. Konzistentně se naopak ukázal úspěch při snaze o zrychlení uložením dvou vektorů do jednoho prokládaného pole, pokud byly použity maximálně 8 bytů velké struktury. Proti očekáváním z předešlého testu se ukázaly v některých případech lepší i 16 bytů velké struktury namísto dvou polí se strukturami o 8 bytech. Potvrzuje se také, že jako zajímavé varianty pro praxi se ukazují hlavně dvojice typů s poloviční přesností, a to jak při ukládání v jednom, tak ve dvou polích. Rovněž platí, že pokud to výpočet umožňuje, je vhodnější ukládat dvě dvojice v jednom poli namísto ukládání po jedné dvojici ve dvou polích.

3.5 Násobení matic

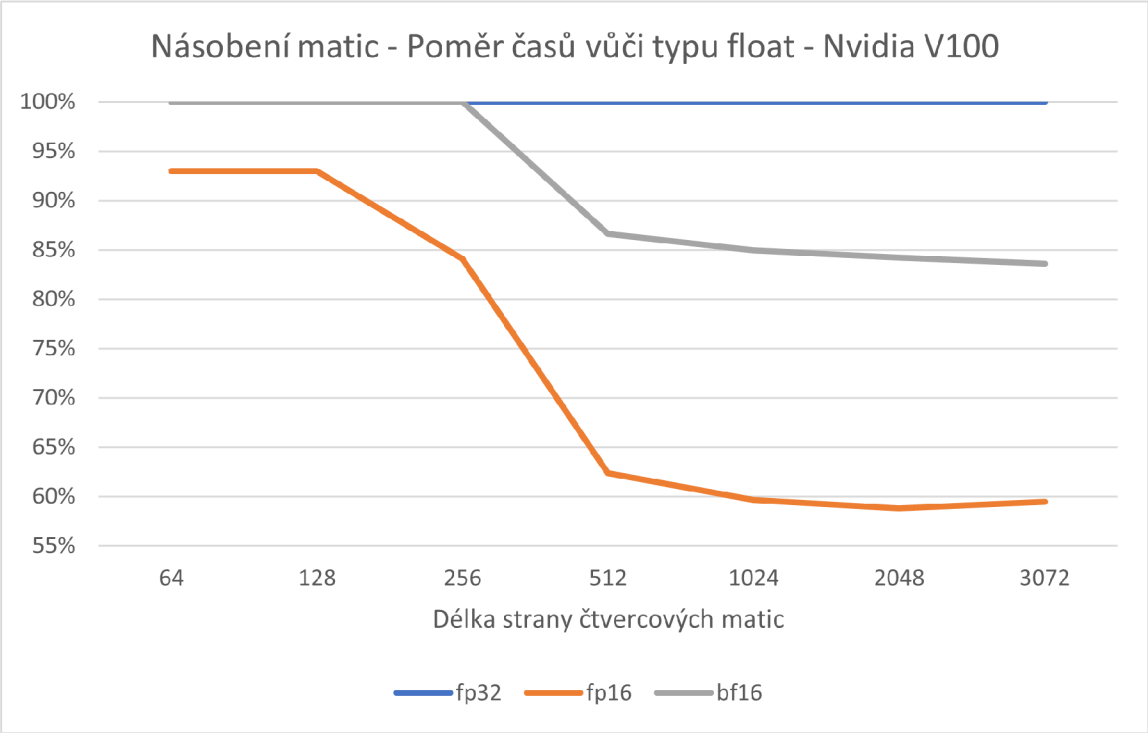
Pro potřeby optimálního násobení matic bylo nutné oproti předchozím testům většího zamyšlení. To spočívalo hlavně v tom, jak přesně využít menších datových typů a jak umožnit vhodné přistupování po dvojicích. Jako ideální pro tento účel se ukázala varianta násobení matic, která již používá optimalizovaný přístup pomocí sdílené paměti. Díky jejímu použití není nutné tak často přistupovat do globální paměti. Sdílená paměť je optimalizovaná na zarovnané přístupy po 4 bytech, lze ji tedy snadno použít také pro přístupy ke dvojicím po 2 bytech. Pro načtení dvojic do sdílené paměti stačí provést úpravu kódu, která je podobná jednomu rozbalení smyčky. Nejedná se tedy o nic technicky složitého. Vzhledem k velikosti bank tento přístup nicméně neumožňuje velké snahy o optimalizace ukládáním po větších strukturách. Nebylo proto možné testovat více různých variant, jako tomu bylo u předcházejících testů. Ve srovnání s předešlými testy je zde také velký rozdíl v tom, že hlavní omezení by nemělo být v propustnosti paměti, nýbrž v reálných výpočtech. Takové optimalizace by tedy stejně neměly vést k citelně velkému nárůstu výkonu, i kdyby byly technicky možné.

Graf 3.11 je poměrně prostý. U malých matic o straně 64 a 128 prvků je trvání na grafické kartě Nvidia A100 takřka konstantní. Z toho důvodu zde ani nelze očekávat velké zrychlení pouhým zmenšením přesnosti čísel. Pro větší matice nicméně dochází k dobrému škálování s vyšším než třetinovým zkrácením času pro čtvercové matice se stranou alespoň 1024 prvků.

Graf 3.12 konečně ukazuje také rozdíl v použití datových typů `brain float` a `half` na grafické kartě Nvidia V100, která, jak bylo již dříve zmíněno, nemá pro relativně nový typ `brain float` podporu na úrovni hardware. Pro malé matice tak lze vidět, že čas je v praxi konstantní, jelikož omezení je hlavně na samotném provádění 32bitových instrukcí násobení. Pro větší matice nicméně dochází i přesto k přibližně 15% zkrácení času čistě na základě zmenšení objemu dat na polovinu. Pro datový typ `half` pak platí, že u malých matic dochází pouze k malému zrychlení díky vektorovému provádění dvou 16bitových instrukcí násobení synchronně. Pro matice s velikostí strany alespoň 512 prvků již však dochází ke zkrácení času o cca 40 %. Jedná se tak o významné zrychlení i ve srovnání s prostým uložením v poloviční přesnosti s výpočty v jednoduché přesnosti.



Obrázek 3.11: Graf Zrychlení výpočtu násobení matic - Nvidia A100



Obrázek 3.12: Graf Zrychlení výpočtu násobení matic - Nvidia V100

Kapitola 4

Analýza programu k-Wave a návrh řešení

V této kapitole se budu věnovat v první části analýze softwarového balíku k-Wave a možnostem jeho zrychlení za využití aritmetiky se sníženou přesností. Pozornost bude věnována návrhu řešení na základě předešlých testů a zjištění, ale také zdůvodnění, čemu se naopak z jakých důvodů nemusí mít smysl věnovat. V druhé části se budu věnovat detailní analýze používaných fyzikálních charakteristik a matic jiných typů dat. Zde bude zhodnoceno, pro která data má, či nemá smysl, pokoušet se o převod do datových typů s nižší přesností a který z dvou možných datových typů s poloviční přesností bude možné či vhodné použít.

4.1 Analýza k-Wave a návrh řešení

Na základě provedených průzkumů a testů v kapitole 3 lze usoudit, že použitím datových typů s nižší přesností lze dosáhnout signifikantních zrychlení programu. Všechny provedené testy toto tvrzení podporují dokonce i v případech, kdy zařízení nemá přímou hardwarovou podporu pro výpočty s daným datovým typem s nižší přesností. V praxi bylo ověřeno, že pro dosažení dobrých zrychlení je ovšem nutné používat minimálně dvojice těchto typů s poloviční přesností. V opačném případě programy trpí na nadměrně sníženou paměťovou propustnost.

V případech, kdy to bude možné, bude preferované použití dvojic typu `half`, který má oproti typu `brain float` vyšší přesnost. Jeho nevýhodou je nicméně řádově nižší rozsah. Jelikož se jedná o fyzikálně založené výpočty, lze ovšem odhadnout, pro které fyzikální veličiny by tento nižší rozsah neměl představovat problém. Použití typu `half` teoreticky může navíc dosahovat vyšších zrychlení na starších platformách generace Volta a Pascal. Použití typu `brain float` by mělo být naopak bezpečné z pohledu rozsahů. To je bonus hlavně tehdy, kdy by nebyla úplná jistota rozsahem veličin. To platí primárně v horní hladině hodnot. Pro příliš vysoké hodnoty by totiž při použití typu `half` došlo k přetečení na hodnotu nekonečno, což by s vysokou pravděpodobností úplně zničilo všechny související výpočty. Nicméně může mít typ `brain float` výhodu i za předpokladu, že se tak vysoké hodnoty nebudou moct vyskytovat, jak totiž popisuji v sekci 2.1, pro typ `half` dochází k zaokrouhlení na hodnotu 0 již při mnohem vyšších hodnotách ve srovnání s typem `brain float`. Ironicky by tak mohla nastat situace, kdy přesnost bude vlivem zaokrouhlování na hodnotu 0 vyšší s méně přesným typem `brain float`. Při implementaci proto může být v některých případech vhodné testovat vliv na přesnost při využití obou typů. Případně by

mohlo potenciálně pomoct vytknutí nějaké hodnoty před zápisem do paměti a provedení inverzní operace při načítání. Taková operace by měla být v datovou propustností omezených částech prakticky zadarmo co se týče času.

V rámci implementace bude nejprve nutné definovat nový přepínač pro zapnutí výpočtů s nižší přesností. Díky tomu by měli mít uživatelé možnost vybrat si mezi lepší přesností a vyšší rychlostí výpočtů. Výpočty v redukované přesnosti případně také umožní spouštění objemnějších simulací v případech, kdy pro výpočty v běžné přesnosti nebude k dispozici dostatečné množství paměti grafické karty. Pro snazší testování vlivů jednotlivých datových konverzí na přesnost by mohlo být navíc praktické implementovat volitelné přepínače pro jednotlivé matice dat.

Dále bude nutné implementovat nové třídy pro matice dat s typy ve snížené přesnosti. Zde bude nutné řešit mimo jiné načítání a konverzi dat, čehož půjde využít pro prvotní fázi testování. Je proto v plánu nejdříve třídy připravit ale implementovat s využitím datového typu `float`, do kterého se pouze budou ukládat data s dopřednou konverzí do snížené přesnosti, což by mělo usnadnit prvotní testování efektů. V pozdější fázi budou matice převedeny do typů `half` a `brain float` a posléze do dvojic těchto typů, což by již však měla být pouze minimální úprava, která by neměla vyžadovat velké zásahy. Dále se bude možné také zamyslet nad potenciálem vyzkoušení optimalizovaného způsobu ukládání dat dvou množin do jednoho prokládaného pole, který byl představen v testech 3.3 a 3.4. Pro takové ukládání jsou vhodnými kandidáty veličiny, které pro heterogenní prostředí mají různé hodnoty v různých směrech. Příkladem může být hustota. Nelze však dopředu odhadnout případný vliv takové optimalizace v mnohem složitější aplikaci. Proto má takový případný pokus jen nízkou prioritu.

Po detailnější analýze softwarového balíku k-Wave jej lze rozdělit na dvě pro tuto práci oddělené části. První částí jsou výpočty využívající Fourierovy transformace. Ty jsou prakticky závislé na datovém typu `float`, a tak se jim v pokusech o optimalizaci pomocí využívání typů se sníženou přesností nebudu věnovat. Z toho ovšem vyplývá také, že matice dat pro tyto výpočty nebude možné transformovat do nižší přesnosti, což omezuje potenciální množinu veličin pro takovou optimalizaci. Druhou částí jsou převážně kernely, u kterých lze vidět, že výpočty jsou obecně vzato omezené hlavně datovou propustností. Nad jednotlivými daty zde probíhá vždy jen nízký počet operací. Díky tomu je nicméně jasné, že lze před samotným prováděním výpočtů data převádět do objemnějšího typu `float`. Jak bylo totiž prakticky vyzkoušeno při testování v kapitole 3, potenciálně dvojnásobná rychlost vektorových aritmetických operací s menšími datovými typy nemá při omezení datovou propustností praktický efekt na celkovou dobu běhu aplikace.

4.2 Analýza vhodnosti dat pro konverzi do nižší přesnosti

Dále je vhodné dopředu analyzovat, které datové typy bude možné převádět do nižší přesnosti a pro která z nich může být vhodné použít datový typ `half`. Použití typu `half` je, jak již bylo zmíněno, možné pouze za předpokladu, že je alespoň přibližně jasná horní hranice dat. Obecně navíc data nesmí být používaná v částech kódu provádějící výpočty Fourierových transformací.

V datech si lze dále povšimnout, že ne všechny matice mají stejně velké dimenze. Pokud se jedná o 1D vektory, tak jejich redukce nepřinese velký efekt pro velkou 3D simulaci, a to ani pro časovou, ani pro prostorovou složitost programu. Podle dimenzí zde lze rozdělit matice dat na 3 typy. Podle nich bude dále analyzováno, zda má smysl dané veličiny snižovat do nižší dimenze.

- Plné dimenze - jedná se o až třídimenzionální data, která jsou obzvlášť v případě 3D simulací velmi objemná.
- Redukované dimenze - tato data mají oproti datům s plnými dimenzemi v ose `X` poloviční velikost, v osách `Y` a `Z` naopak velikost kopírují. Celkový objem dat je proto ve srovnání s plnými dimenzemi poloviční.
- Jednodimenzionální data - Tato data kopírují vždy jednu z plných dimenzí. Obzvlášť pro 3D simulace je tak jejich velikost relativně malá a oproti předchozím dvěma typům zanedbatelná.

Potenciálně vhodné veličiny pro konverzi do nižší přesnosti

V této části práce bude analyzován výčet veličin, které je potenciálně možné a vhodné převést do nižší přesnosti.

Akustický koeficient nelinearity

První maticí dat, kterou je možné převést do nižší přesnosti, je akustický koeficient nelinearity. Tento koeficient je vyjadřovaný jako poměr B/A, v kódu jej lze proto nalézt pod zkratkou `BOnA`. Jedná se o jednu z matic v plných dimenzích, redukce by proto měla přinést minimálně prostorovou úsporu.

Pro koeficient dále platí, že by mělo být v pořádku používat datový typ `half` s nižší přesností. To lze odvodit z knihy [22], ve které se nachází vyčíslené hodnoty tohoto koeficientu pro řadu materiálů. Nejvyšší hodnota je přitom pouze 11,54. Takový rozsah by bylo teoreticky možné ukládat i do přesnějšího z osmibitových typů, které byly zmíněny v kapitole 2.1.4. Je tedy možné v budoucnu provést další optimalizaci tímto směrem.

Rychlost zvuku

Další vhodnou veličinou pro ukládání v nižší přesnosti by měla být rychlost zvuku. Opět se jedná o jednu z větších matic s plnými dimenzemi.

Pro rychlost zvuku lze snadno dohledat mnoho příkladů pro plynné, tekuté i pevné materiály. Například z [3] tak lze snadno vypozorovat, že rychlost zvuku v žádném běžném médiu nepřesahuje řádově jednotky tisíc metrů za sekundu. I v tomto případě je tedy možné použití typu `half`. Zároveň tato rychlost bude zřejmě vždy alespoň ve stovkách. Z toho důvodu se není třeba bát zaokrouhlování na hodnotu 0 při příliš malých hodnotách. Použití typu `brain float` by tedy v tomto případě nemělo mít v žádném případě výhodu ve srovnání s typem `half`. Vzhledem k těmto silným omezením na rozsah rychlosti zvuku by teoreticky mohlo být možné přesnost dále zvýšit pomocí vytknutí nějaké konstanty a v budoucnu by proto mělo být možné provést případnou další redukci do některého z osmibitových typů.

Hustota

Jako potenciálně vhodná charakteristika pro redukci přesnosti se jeví i hustota. Ta je zde reprezentována dokonce až čtyřmi maticemi s plnými dimenzemi. Jedna z nich je matice počáteční hustoty, dalšími jsou matice akustické hustoty posunuté v různých osách. Převedením do nižší přesnosti proto lze očekávat razantní redukci v celkovém objemu dat.

Co se týče rozsahů této veličiny, nejlehčím prvkem na naší planetě je vodík. Jak se lze dočíst z [1], vodík má hustotu 0,08375 kg/m^3 za standardní teploty a tlaku. Mezi nejtěžší materiály patří naopak osmium a iridium. Ty mají dle [8] za stejných podmínek hustotu 22590 a 22560 kg/m^3. Maximální hodnota datového typu `half` je tedy téměř třikrát vyšší. Lze se proto domnívat, že v blízké budoucnosti nebudou vytvořeny ani umělé prvky, které by se staly běžně využívanými a pro které by byla nutná podpora a zároveň by byly mimo limity datového typu `half`. Vzhledem k plánu přepínače pro zapnutí výpočtů s nižší přesností by však i poté bylo možné v takových okrajových situacích s těžšími materiály počítat simulace pomaleji ve vyšší přesnosti. Nejednalo by se tedy o natolik zásadní nedostatek.

Podíl časového kroku a počáteční hustoty posunuté v různých osách

V tomto případě se jedná o tři předpočítané matice podílů s plnými dimenzemi. Od snížené přesnosti tedy lze očekávat minimálně nezanedbatelný efekt na množství potřebné paměti.

Důležité je, že jak již bylo dříve zmíněno, hustotu by mělo být možné bez problémů vyjádřit pomocí datového typu `half`. Na délku časového kroku se vedle toho lze dívat jako na konstantu. Podíl konstanty a tlaku by tedy rovněž neměl být problém vyjádřit pomocí typu `half`. V tomto tvrzení lze vycházet z toho, že maximální záporný a maximální kladný exponent typu `half` si jsou v absolutní hodnotě rovny.

Zdroj počátečního tlaku

Zdroj počátečního tlaku je aplikaci předán jako jedna matice s plnými dimenzemi.

Narozdíl od většiny ostatních veličin je tato veličina používána jen minimálně. Dá se proto předpokládat, že její optimalizace bude mít pouze prostorový význam, nikoliv však časový. Navíc by ani použití typu `brain float` s nižší přesností nemělo hrát velkou roli na cílovou přesnost.

Zdroj tlakových vln

Zdroj tlakových vln je vstup, jehož matice je v aktuální implementaci unikátně odlišná při porovnání s ostatními maticemi. Je vyjádřená 2D maticí, kde jednou dimenzí je čas. Druhou dimenzí je velikost masky, tedy potenciálně až součin dimenzí plné matice. Data proto mohou být zásadně objemnější než všechny ostatní matice. To bude nicméně řešeno v blízké budoucnosti postupným bufferováním.

Tlak nemá žádným způsobem jasně omezené hranice, proto se použití typu `half` nezdá v tomto případě jako bezpečné. Při této redukci nelze očekávat velké zrychlení kódu, jelikož jsou data používána jen minimálně. Lze nicméně očekávat značnou prostorovou redukci v některých typech simulací.

Absorpční koeficienty Tau a Eta

Absorpční koeficienty tau a eta jsou reprezentovány každý vlastní maticí s plnými dimenzemi.

Data jsou v tomto případě počítána až po startu aplikace a odhad rozsahů a typických hodnot není tak snadný. Bude tedy záležitostí experimentů, zda půjde použít datový typ `half`, nebo bude vhodnější použití datového typu `brain float`.

Absorpční koeficienty Nabla

Absorpční koeficienty nabla jsou dvě matice koeficientů s redukovanými dimenzemi. V případě úspěšné redukce těchto matic by proto mohly být zároveň kandidátem pro prokládané ukládání hodnot po dvojicích.

Obdobně jako u absorpčních koeficientů tau a eta se zde bohužel nedá snadno odhadnout nějaký typický rozsah hodnot. Který datový typ bude vhodnější použít bude proto opět záležitost experimentů.

Kappa koeficient

Kappa koeficient je veličina vyjádřená jednou maticí s redukovanými dimenzemi. V porovnání s předchozími veličinami tak má nižší prioritu pro převod do nižší přesnosti, protože efekt na paměť bude jen minimální.

Obdobně jako u předchozích koeficientů se nezdá, že by se dal možný rozsah hodnot jednoduše analyzovat. Bude tedy nutné obě varianty otestovat a poté teprve provést finální rozhodnutí.

Kappa koeficient zdroje

Kappa koeficient zdroje je poslední nalezenou veličinou, která se jeví jako potenciálně vhodná pro převod do redukované přesnosti. Obdobně jako kappa koeficient je dána jednou maticí s redukovanými dimenzemi, takže má jen nízkou prioritu pro redukci při implementační fázi.

Rozsah hodnot je v tomto případě velmi jasný, jelikož je hodnota spočítána jako kosinus. Z toho plyne pevně daný rozsah hodnot v intervalu od -1 do 1. Jako vhodnější se tak jeví formát `half`. Pro zvýšení přesnosti může být také vhodné z této veličiny vytknout nějakou hodnotu.

Nevhodné veličiny pro převod do snížené přesnosti

Následující seznam veličin by pravděpodobně nebylo vhodné převádět do nižší přesnosti. Důvodem je, že redukce by nepřinesla příliš efekt, jelikož se jedná pouze o jednodimenzionální data, která jsou ve srovnání s jinými vícedimenzionálními daty zanedbatelně malá. Nebylo by tak docíleno prostorové úspory a nejspíše ani časové.

- PML (Perfectly matched layer) - tři matice s jednoduchou dimenzí
- PML na posunuté mřížce v různých osách - tři matice s jednoduchou dimenzí
- vstupní data zdroje - jedna matice s jednoduchou dimenzí
- vstupní data akustické rychlosti zdroje v různých směrech - tři matice s jednoduchou dimenzí
- neuniformní mřížky akustické rychlosti v různých směrech - tři matice s jednoduchou dimenzí
- posunuté neuniformní mřížky akustické rychlosti v různých směrech - tři matice s jednoduchou dimenzí

Nekonvertovatelné veličiny do snížené přesnosti

Následující výčet veličin není prakticky převeditelný do nižší přesnosti z důvodu využití ve výpočtech Fourierových transformací.

- tlak - jedna matice s plnými dimenzemi
- akustická akcelerace v různých osách - tři matice s plnými dimenzemi
- rychlost na posunuté mřížce v různých osách - tři matice s plnými dimenzemi
- Fourierův posun v různých osách - šest komplexních maticí s jednoduchou dimenzí
- rychlost posunutá v různých osách - tři matice s plnými dimenzemi
- dočasné matice pro různé účely - tři matice s plnými dimenzemi
- Negativně posunutá rychlost v různých osách - tři komplexní matice s jednoduchou dimenzí
- dočasné matice pro Fourierovy transformace - čtyři komplexní matice s redukovanými dimenzemi

Kromě výše jmenovaných množin dat se v kódu nachází několik indexovacích matic, které ze své povahy logicky nelze převádět do nižší přesnosti.

Kapitola 5

Implementace

V rámci této diplomové práce vzniklo řešení pro akceleraci softwarového balíku k-Wave implementovaného pomocí platformy CUDA pro grafické karty společnosti Nvidia za využití aritmetiky se sníženou přesností. Tato kapitola se bude věnovat implementačním detailům vytvořeného řešení a popisu, jak jsem při daném řešení postupoval. Pozornost bude věnována zdůvodnění některých změn ve srovnání s původní implementací a problémům, se kterými jsem se při implementaci potýkal. Dále uvedu, které návrhy z kapitoly 4.1 se podařilo splnit, které ne a případně, zda jsou některé nerealizované návrhy realizovatelné a do budoucna hodná doporučení realizovat je.

5.1 Přepínač pro redukovanou přesnost

Prvním krokem při implementaci bylo dát uživateli možnost používat aplikaci v módu s redukovanou přesností a plnou přesností dle vlastní volby. Pro tuto volbu byl do aplikace přidán volitelný přepínač `--reduced_precision`.

Pro zprovoznění toho přepínače bylo nutné ve všech relevantních šablonách, které se v aplikaci využívají pro optimalizaci, přidat nový parametr s typem pravdivostní hodnoty. Zde jsem se setkal nejprve s problémem nesprávného spouštění šablon. To bylo způsobeno tím, že v kódu jsou zadefinovány pro některé části implicitní hodnoty parametrů, jelikož ne ve všech částech jsou všechny přepínače relevantní. Do šablon byl tento nový parametr nejprve přidán jako poslední. Kvůli tomu bylo nicméně snadné přehlédnout velké množství míst volání, kde měl být tento nový parametr předán. Z toho důvodu jsem se posléze rozhodl změnit jejich pořadí a tento nový parametr předávat v těchto funkcích s předdefinovanými implicitními hodnotami jako první. To vedlo k jasnému zobrazení chyb při překladu, jelikož poté typově neseděly parametry nově zvolenému pořadí. Tento přístup umožnil snadno nalézt všechny oblasti v kódu, kde bylo nutné použít tento nový parametr pro redukovanou přesnost, díky tomu se rovněž snížila možnost zavedení nechtěných chyb v rozsáhlém kódu, který bylo nutné najednou měnit na mnoha místech.

Implicitní hodnotou tohoto nového přepínače je pravdivostní hodnota `false`. Tím se odlišuje od ostatních pravdivostních parametrů, které jsou používány v šablonách a na základě kterých se spouští správná varianta šablon funkcí. Ty jsou naopak všechny definovány s implicitní hodnotou `true`. Toto rozhodnutí bylo učiněno, jelikož logickou implicitní volbou by měla být plná přesnost. Pro aktuální implementaci toto rozhodnutí však nemá praktický efekt, jelikož z důvodu pořadí parametrů musí být tento parametr u všech volání předán explicitně.

Ačkoliv jsem v kapitole 4.1 navrhoval možnost implementace volitelných přepínačů pro jednotlivé matice, tato volba se ukázala netriviální a prakticky neaplikovatelnou ve velkém měřítku. Znamenalo by to vysoké množství nově předávaných parametrů a úprav při každém přidaném či odebraném parametru. Navíc by v kódu bylo nepatřičně velké množství větvení dle těchto parametrů. Od tohoto návrhu jsem proto musel upustit.

Oproti návrhu byla navíc implementována použití dvou dalších parametrů příkazové řádky. Prvním z těchto parametrů je `--reduced_precision_extra`. Tímto parametrem je zapnuta pokročilá volba extra optimalizace. Při ní budou do nižší přesnosti převedeny některé další matice, které nicméně mají statisticky větší vliv na chybovost výsledků a jejich pozitiva naopak nemají příliš velký efekt. Druhým přidaným parametrem příkazové řádky je `--reduced_precision_min`. Tento parametr má přesně opačný efekt. To znamená, že ve srovnání se základní volbou `--reduced_precision`, která byla implementována jako první, část optimalizací vypíná. Díky tomu lze dosáhnout lepších výsledků s minimální chybovostí výpočtů. Konkrétní rozdíly v tom, které veličiny byly zařazeny do které úrovně optimalizací lze nalézt v sekci 5.4. Míru rozdílů v prostorové náročnosti, časové náročnosti a přesnosti výpočtů mezi těmito třemi variantami spuštění lze nalézt podrobně vyhodnocenou dále v kapitole 6.

5.2 Třídy pro matice dat s redukovanou přesností

Pro použití matic dat využívajících typy s redukovanou přesností bylo nutné vytvořit nové třídy. Jedná se o třídy `BaseHalfMatrix`, `HalfMatrix`, `BaseBfloatMatrix` a `BfloatMatrix`. Při jejich implementaci jsem vycházel z již existujících tříd `BaseFloatMatrix` a `RealMatrix`. Jejich celou implementaci lze nalézt v souborech znázorněných v následující souborové struktuře.

```
k-Wave-Fluid-CUDA
└── MatrixClasses
    ├── BaseBfloatMatrix.cpp
    ├── BaseBfloatMatrix.h
    ├── BaseHalfMatrix.cpp
    ├── BaseHalfMatrix.h
    ├── BfloatMatrix.cpp
    ├── BfloatMatrix.h
    ├── HalfMatrix.cpp
    └── HalfMatrix.h
```

Jak bylo uvedeno při návrhu v předešlé kapitole 4.1, nejprve jsem ponechal implementace těchto nových tříd nad datovým typem `float`. Po zakomponování potřebných referencí na nové třídy v kódu a do souboru `Makefile` tak byla implementace úspěšně zprovozněna s využitím nových tříd. Dále se bylo možné přesunout k úpravám těchto tříd. V jejich provedení se nacházelo velké množství explicitního použití datového typu `float`, či jeho velikosti za pomocí operátoru `sizeof` a v některých případech bylo potažmo přímo počítáno s jeho velikostí 4 byty. Pro budoucí snazší migraci dat do nižší přesnosti jsem se rozhodl tyto části v nových třídách upravit. V současné podobě je použitý datový typ jmenován explicitně pouze v makru na jednom místě v souborech `BaseHalfMatrix.h` a `BaseBfloatMatrix.h`. Operátor `sizeof` dále využívá ukazatele na data matic. Ukázku tohoto přístupu lze vidět v následující ukázce kódu 5.1 pro alokaci matic typu `half`.

```
void BaseHalfMatrix::allocateMemory()
{
    // Size of memory to allocate
    size_t sizeInBytes = mCapacity * sizeof(*mHostData);

    // Allocate CPU memory
    mHostData = static_cast<TYPE*>(_mm_malloc(sizeInBytes, kDataAlignment));
    if (!mHostData)
        throw std::bad_alloc();

    // Register Host memory (pin in memory)
    cudaHostRegister(mHostData, sizeInBytes, cudaHostRegisterPortable);

    // Allocate memory on the GPU
    if ((cudaMalloc<TYPE>(&mDeviceData, sizeInBytes) != cudaSuccess))
        throw std::bad_alloc();

    zeroMatrix();
}// end of allocateMemory
```

Výpis 5.1: Alokace paměti pro matici dat typu half

Výše uvedené principy změn umožnily později snazší úpravy při konverzi na datové typy `half` a `__nv_bfloat16` a následně také při změnách pro využívání dvojic těchto typů. V případě budoucí snahy využít osmibitové typy zmíněné v sekci 2.1.4 bych proto doporučil vycházet z těchto nových tříd. Jejich úpravy pro práci s novými datovými typy by měly být signifikantně snadnější ve srovnání náročností úprav již dříve existujících tříd.

5.3 Gettery, kernely a související funkce

Zde bude věnována pozornost jmenování nových funkcí a popisu důležitých úprav v původních funkcích, které byly pro potřeby této práce implementovány.

Gettery

Pro možnost práce s novými datovými typy bylo nutné vytvořit nové gettery, neboli funkce na získání potřebných hodnot a ukazatelů nejen na nové typy matic. Pro nové třídy byly implementovány gettery `getHalfMatrix` a `getBfloatMatrix`. Tyto gettery vrací z kontejneru ukazatel na nově implementované objekty třídy `HalfMatrix`, respektive `BfloatMatrix`.

Dále byly implementovány funkce `getHalfData` a `getBfloatData`. Ty vrací daná data jako ukazatel typu `half`, respektive `__nv_bfloat16`. Pro třídu `KSpaceFirstOrderSolver` se jedná o ukazatel na CPU verzi dat a v případě třídy SolverCudaKernels jde o ukazatel na GPU verzi. Pro implementaci je důležité, že ačkoliv jsou hodnoty ukládány po dvojicích, v paměti jsou data zarovnána stejně, jako kdyby se o dvojice nejednalo. Díky tomu je možné provádět přetypování z ukazatele na dvojice těchto typů a gettery tak nebyly při vývoji závislé na tom, zda byla aktuální verze matic implementována pomocí dvojic. K těmto getterům byly následně implementovány alternativní gettery `getHalf2Data` a `getBfloat2Data`, které umožňují získat ukazatel na data jako na dvojice hodnot. Jak bylo totiž vysvětleno

v kapitole 2.2 a prakticky ověřeno v kapitole 3, pro efektivní práci je pro typy s redukovanou přesností důležité přistupovat k datům alespoň po dvojicích hodnot. To samozřejmě není vždy možné, proto je nutné zachovat také možnost přístupu po jednotlivých hodnotách.

Pro možnost konzistentní práce s dvojicemi dat bylo v cíli nutné přidat také getter `getReal2Data` pro získání ukazatele na dvojice hodnot typu `float2`. Obdobně jako u ukazatelů na dvojice nebo jednotlivé hodnoty u typů `half` a `__nv_bfloat16` je zde využíváno faktu, že zarovnání dat je pro dvojice i jednotlivé hodnoty dat totožné. Lze tedy provádět přetypování i tímto opačným směrem z ukazatele na `float` na ukazatel na dvojice hodnot.

Mimo getterů pro třídy matic a jejich data byly dále přidány gettery pro nové argumenty příkazové řádky. Konkrétně se jedná o funkce `isReducedPrecisionMinEnabled`, `isReducedPrecisionEnabled` a `isReducedPrecisionExtraEnabled`. Díky nim je možné se v kódu rozhodovat, které šablony a úseky funkcí spouštět pro tři různé úrovně redukované přesnosti a pro plnou přesnost výpočtů.

Kernely a související funkce

Hlavní změnou v implementaci byla nutnost přidání nového parametru do šablon kernelů, které využívají některou z veličin, jež byly převedeny do nižší přesnosti. Na základě tohoto parametru byla nejprve pro tyto kernely aplikována jedna podmínka, která dané funkce rozděluje na dvě oddělené funkční části pro redukovanou přesnost a pro plnou přesnost výpočtů. K tomuto provedení jsem se rozhodl, jelikož z redukované přesnosti plyne používání jiných datových typů u mnoha veličin, nebo případně stále alespoň používání dvojic typu `float`. K těm se dále váží navíc další odlišné funkce a gettery. Ukázku popsaných principů lze vidět v následující ukázce kernelu 5.2.

```
template<bool reducedPrecisionFlag>
__global__ void cudaComputeAbsorptionTerm()
{
    cuFloatComplex* fftPart1 = getComplexData(kTempCufftX);
    cuFloatComplex* fftPart2 = getComplexData(kTempCufftY);

    if (reducedPrecisionFlag) {
        const __nv_bfloat162* absorbNabla = getBfloat2Data(kAbsorbNabla1);
        for (auto i = getIndex(); i < nElementsComplex; i += getStride()) {
            const __nv_bfloat162 nabla = absorbNabla[i];
            fftPart1[i] *= nabla.x;
            fftPart2[i] *= nabla.y;
        }
    } else {
        const float* absorbNabla1 = getRealData(kAbsorbNabla1);
        const float* absorbNabla2 = getRealData(kAbsorbNabla2);
        for (auto i = getIndex(); i < nElementsComplex; i += getStride()) {
            fftPart1[i] *= absorbNabla1[i];
            fftPart2[i] *= absorbNabla2[i];
        }
    }
}// end of cudaComputeAbsorptionTerm
```

Výpis 5.2: CUDA kernel pro výpočet absorpčního členu pomocí nabla koeficientů

Alternativně k dříve popsanému a zobrazenému způsobu implementace by v šablonách šlo na základě podmínek odvozovat typy a dle typů odvozovat, kterou funkci použít. Tento přístup se však zdál být implementačně příliš složitý, a to primárně z důvodu nutnosti jednotného způsobu zápisu výpočtů. Napsat jednotný zápis tak, aby byl kompatibilní pro různé datové typy, se přitom ukázal v některých případech netriviální i pro snadné testovací aplikace, které byly vytvořeny pro kapitolu 3.

Dále byly implementovány možnosti využití minimalizované a rozšířené redukce dat dle vznikající chybovosti výpočtů. Proto lze u některých kernelů nalézt v šablonách další parametr pro minimalizovanou redukci, tyto kernely jsou tedy rozdělené dokonce na tři samostatné části. Jejich rozhraní nicméně nešlo jednoduše přidat další parametr do šablony bez rozsáhlých úprav. Proto je tato volba předávaná jako argument funkce, jak lze vidět v ukázce 5.3. V případě rozšířené redukce nedošlo k žádnému konfliktu, kdy by některý kernel měl tři různé varianty chování. Z toho důvodu byl v patřičných rozhraních pouze nahrazen parametr pro redukovanou přesnost za parametr pro extra redukovanou přesnost.

```
template<bool reducedPrecisionFlag,
         SD dimension,
         bool rho0ScalarFlag,
         bool bOnAScalarFlag,
         bool c0ScalarFlag,
         bool alphaCoefScalarFlag>
void SolverCudaKernels<reducedPrecisionFlag,
                       dimension,
                       rho0ScalarFlag,
                       bOnAScalarFlag,
                       c0ScalarFlag,
                       alphaCoefScalarFlag>::
addPressureScaledSource(const RealMatrix& scaledSource,
                        bool reducedPrecisionMinFlag)
{
    if (reducedPrecisionMinFlag) {
        cudaAddPressureScaledSource<dimension, false, true>
                      <<<getSolverGridSize1D(), getSolverBlockSize1D()>>>
                      (scaledSource.getDeviceData());
    } else {
        cudaAddPressureScaledSource<dimension, reducedPrecisionFlag, false>
                      <<<getSolverGridSize1D(), getSolverBlockSize1D()>>>
                      (scaledSource.getDeviceData());
    }

    // Check for errors
    cudaCheckErrors(cudaGetLastError());
}// end of addPressureScaledSource
```

Výpis 5.3: Rozhraní kernelu pro přidání škálovaného zdroje tlaku k akustické hustotě

U kernelu `cudaInsertSourceIntoScalingMatrix` v ukázce 5.4 lze dále nalézt principiálně odlišný přístup. V tomto případě je šablona využívána k odvození typu, namísto k rozhodování, zda se má spustit verze s redukovanou přesností. V tomto případě šlo využít

toto řešení efektivněji, protože jediná veličina, pro kterou je nutné řešit odlišné chování, je funkci předávaná jako argument. Tato funkce navíc neumožňuje jednoduchý přístup k datům po dvojicích hodnot, a tak lze jednoduše data převést v každém případě na typ `float`. Tomuto kernelu slouží jako rozhraní ke spouštění funkce `insertSourceIntoScalingMatrix`. I tomuto rozhraní se ona veličina předává pomocí argumentu. V tomto případě je však již pevně definovaná šablona s parametry pro funkce spadající do třídy `SolverCudaKernels`. Proto nešlo vyřešit variabilitu typu této matice pomocí parametru šablony. Z toho důvodu byla pro výpočty s redukovanou přesností zavedená druhá varianta tohoto rozhraní, která se liší použitým typem matice.

```
template<bool isMany, typename matrixType>
__global__ void cudaInsertSourceIntoScalingMatrix(float* scaledSource,
                                                  const matrixType* sourceInput,
                                                  const size_t* sourceIndex,
                                                  const size_t sourceSize,
                                                  const size_t timeIndex)
{
    // Set 1D or 2D step for source
    const auto step = (isMany) ? timeIndex * sourceSize : timeIndex;

    // Different pressure sources
    if (isMany) {
        // Multiple signals
        for (auto i = getIndex(); i < sourceSize; i += getStride()) {
            scaledSource[sourceIndex[i]] = float(sourceInput[step + i]);
        }
    } else {
        // Single signal
        for (auto i = getIndex(); i < sourceSize; i += getStride()) {
            scaledSource[sourceIndex[i]] = float(sourceInput[step]);
        }
    }
}// end of cudaInsertSourceIntoScalingMatrix
```

Výpis 5.4: CUDA kernel pro přidání zdroje tlaku do akustické hustoty

Předávání veličin jako ukazatel na matici dat skrze parametry lze nalézt také v některých dalších kernelech. V takových případech je typicky pro redukovanou verzi využita možnost explicitního přetypování ukazatele typu `float` na ukazatel na typ `float2`.

V některých případech lze v kernelech nalézt upravený zápis výpočtu pro verzi se sníženou přesností ve srovnání se zápisem pro výpočty s plnou přesností. Hlavní příčinou je obvykle využívání dvojic hodnot. Platforma `CUDA` totiž nepodporuje veškeré operace nad datovým typem `float2`, které ovšem byly původně využity nad datovým typem `float`. Takovou operací je například operace odčítání. Při práci s typem `float2` tak bylo nutné započítat nejprve zápornou hodnotu přímo do některé proměnné a následně provádět operaci sčítání. Mimo problémy s nepodporovanými operacemi mohly být některé rovnice při práci s dvojicemi hodnot přepsány pro dosažení lepší čitelnosti či efektivity čtení při využití dvojic. Pro pár veličin byl případně také změněn způsob ukládání při výpočtech s redukovanou přesností, což bude detailně probráno v kapitole 5.4.

5.4 Redukované veličiny a související úpravy

V této sekci se budu věnovat jednotlivým veličinám, které byly zmíněny v sekci 4.2 jako teoreticky vhodné pro redukci přesnosti. Pro jednotlivé veličiny bude upřesněno, zda se redukce setkala s úspěchem a do které kategorie optimalizací byla na základě vlivu na přesnost zařazena. Dále bude uvedeno, který datový typ byl nakonec zvolen a případné nutné úpravy kernelů a jiných funkcí ve srovnání s referenční implementací v plné přesnosti.

Matice, které byly pro redukci do nižší přesnosti vybrány jsou vždy buď v redukovaných, nebo v plných dimenzích, přičemž matice s redukovanými dimenzemi je přibližně poloviční velikosti ve srovnání s maticí s plnými dimenzemi. Na základě tohoto bych zde chtěl zadefinovat abstraktní jednotku dat vyjadřující, kolik paměti se podařilo převodem dané veličiny ušetřit. Tu si lze zadefinovat jako 1 jednotka paměti za matici s redukovanými dimenzemi a 2 jednotky paměti za matici s plnými dimenzemi. Na konci kapitoly lze nalézt tabulku 5.1 se souhrnem provedených redukcí pro každou úroveň implementované redukované přesnosti.

Akustický koeficient nelinearity

Akustický koeficient nelinearity, vyjadřovaný jako poměr B/A, se podařilo dle návrhu bez větších obtíží převést na datový typ `half`, který má ve srovnání s typem `brain float` vyšší přesnost. Celkový efekt na přesnost výpočtu této veličiny se jeví jako minimální. Proto je tato veličina převedena do nižší přesnosti již při minimální implementované úrovni výpočtů s redukovanou přesností, tedy při všech třech variantách přepínačů pro redukovanou přesnost, které byly zmíněny v sekci 5.1 dříve. Jedná se tak o jednu matici s plnými dimenzemi a byly tím ušetřeny 2 jednotky paměti.

Rychlost zvuku

Rychlost zvuku byla na převod ve srovnání s ostatními veličinami složitější. V návrhu jsem tuto veličinu zmiňoval jako teoreticky vhodnou pro využití typu `half` vzhledem k jejímu jasnému rozsahu. V kernelech se nicméně využívá předpočítaný kvadrát této rychlosti. Z toho plyne, že cílový rozsah této veličiny byl mimo limity dané typem `half`. Proto byla implementace nejdříve provedena nad datovým typem `brain float`. Pro zvýšení přesnosti byla posléze upravena implementace tak, aby se rychlost nejprve ukládala jako datový typ `half` a až při výpočtu kvadrátů poté jako typ `brain float`. Tato implementace však byla nakonec opuštěna, jelikož z testů vyplynulo, že vhodnější přístup je nemít kvadráty předpočítané a mít data uložená po celou dobu běhu programu pomocí datového typu `half`. Počítání kvadrátů rychlosti při využití totiž zvyšuje přesnost výpočtů o jedno chybějící zaokrouhlení na typ `brain float`. Přitom opakované výpočty při použití v kernelech nemají měřitelný negativní efekt na rychlost běhu programu, jelikož jsou dané oblasti silně omezeny propustností paměti, nikoliv však výpočetním výkonem.

Efekt na přesnost výpočtu je díky výše zmíněným optimalizacím v cíli zanedbatelný. Jedná se proto o další veličinu, jejíž redukce do nižší přesnosti je aktivní již při použití parametru `--reduced_precision_min`. V cíli tak byla tato veličina konzistentně s návrhem převedena na typ `half` a na jedné matici s plnými dimenzemi byly ušetřeny další 2 jednotky paměti.

Hustota

V případě hustoty se jedná o čtyři matice s plnými dimenzemi, přičemž nejprve se budu věnovat počáteční hustotě o jedné matici a poté až třem maticím akustické hustoty posunuté v různých osách.

Jak již bylo zmíněno v návrhu, hustotu lze principiálně uložit do datového typu `half` s nižším rozsahem a lepší přesností. Jelikož hodnoty typicky nenabývají dokonce ani příliš nízkého rozsahu, kde by docházelo k zaokrouhlení na hodnotu 0, je datový typ `half` ideální a bez velkých negativních efektů na přesnost výpočtu. Tato redukce proto byla implementována již od nejnižší úrovně zapnutých prostorových optimalizací. Jedná se tedy o ušetřené 2 jednotky paměti.

Pro akustickou hustotu posunutou v různých osách byl rovněž zvolen typ `half` a bylo prakticky otestováno, že použití datového typu `brain float` vede na mnohem větší chybovost. V případě těchto matic hodnot však dochází k jejich opakovanému přepočítávání, a tak dochází k signifikantně většímu kumulování chyb. Ač se chybovost nezdála statisticky zásadní při implementaci, při detailnějších testech prováděných ke konci vývoje byl prokázán opak. Z toho důvodu byl naimplementován dodatečný přepínač `--reduced_precision_min`, kterým lze zapnout minimalizovanější verzi optimalizací. Při té dochází v praxi k vypnutí optimalizace těchto tří matic posunuté akustické hustoty. Jelikož se v tomto případě jedná o tři matice s plnými dimenzemi, bylo zde dosáhnuto 6 ušetřených jednotek paměti.

Celkově bylo pro na těchto veličinách ušetřeno 8 jednotek paměti pro střední a rozšířenou redukci přesnosti. Pro minimalizovanou redukci byly ušetřeny pouze 2 jednotky paměti.

Podíl časového kroku a počáteční hustoty posunuté v různých osách

V tomto případě se jedná o tři matice dat s plnými dimenzemi. Jak bylo zmíněno v návrhu, je platné, že pro tyto předpočítané matice je vyhovující datový typ `half` a použití datového typu `brain float` přináší signifikantně větší chybovost.

I přes užití přesnějšího typu `half` však byla výsledná chyba výpočtů velmi vysoká. Ve snaze snížit tuto chybovost je proto pro redukovanou přesnost vytknuta delta času ven z předpočítaných matic a je započítávána explicitně v místech užití. Díky tomu se podařilo chybu signifikantně snížit do relativně rozumných mezí. Stejně jako v případě rychlosti šíření zvuku také platí, že jsou tyto části silně omezené datovou propustností, nikoliv výpočetním výkonem. Redundance výpočtů tedy opět nepřináší žádná měřitelná negativa.

Chybovost výsledků je i přes úspěšně snahy o jejich vylepšení nicméně stále řádově větší i ve srovnání s chybou vznikající na opakovaně přepočítávaných maticích posunutých akustických hustot. Jedná se proto o důvod vzniku přepínače `--reduced_precision_extra`. Tato optimalizace je aplikována pouze při použití tohoto rozšířeného přepínače. Zásadní vliv tohoto přepínače na přesnost výpočtů lze pozorovat níže v kapitole 6. Pro tuto rozšířenou možnost se nicméně jedná o dalších 6 jednotek ušetřeného místa.

Zdroj počátečního tlaku

V případě zdroje počátečního tlaku byla optimalizována jedna matice s plnými dimenzemi. Pro redukci byl dle návrhu zvolen datový typ `brain float`. Dle předpokladu se také nepodařilo s tímto méně přesným typem naměřit reálné snížení přesnosti výpočtů. To lze odůvodnit tím, že je tato veličina v testovaných simulacích příliš insignifikantní.

Úspěšně byly tedy ušetřeny 2 jednotky paměti i pro nejnižší míru optimalizací.

Zdroj tlakových vln

Pro zdroj tlakových vln byla zvolena optimalizace s využitím datového typu `brain float`. Platí přitom, jak je zmíněno v návrhu, že se jedná o jednu matici, která je v aktuální implementaci unikátní a může nabývat technicky čtyřrozměrných rozměrů s přidanou osou času. Z toho důvodu by v aktuální implementaci nemožné jednoduše vyjádřit prostorovou optimalizaci. V blízké budoucnosti má však být nasazen update programu k-Wave, kde je tento problém řešen postupným načítáním dat. Lze tedy do budoucna v praxi počítat s jednou maticí dat s plnými dimenzemi.

Efekt na přesnost byl vyhodnocen jako minimální, tato optimalizace je tedy aktivní při všech úrovních redukované přesnosti. Ač v aktuální implementaci může tato optimalizace snižovat potřebné množství paměti téměř na polovinu, do budoucna lze počítat s ušetřenými dvěma jednotkami prostoru.

Absorpční koeficienty Tau a Eta

Absorpční koeficienty jsou reprezentovány dvěma maticemi s plnými dimenzemi. Při jejich redukci byl prvně zvolen typ `brain float`, jelikož rozsahy nejsou stoprocentně jasné. Z testů posléze vyplynulo, že použití typu `half` v tomto případě vede na mnohem vyšší nepřesnost. Ta je nejspíše způsobená příliš malými hodnotami, které se v důsledku při použití typu `half` zaokrouhlují na 0. Relativně zanedbatelné chyba při použití typu `brain float` lze přisoudit tomu, že jsou koeficienty spočítány jednorázově pro celý běh aplikace, díky čemuž nedochází k významnému kumulování chyby.

Úspěšně tak bylo docíleno redukce obou koeficientů do snížené přesnosti. Pro všechny varianty spuštění se sníženou přesností tak lze tak počítat s dalšími čtyřmi ušetřenými jednotkami místa.

Absorpční koeficienty Nabla

V případě absorpčních koeficientů nabla se jedná o dvě matice s redukovanými dimenzemi. Obdobně jako v případě koeficientů tau a eta není jejich rozsah stoprocentně jasný. Proto byl pro výslednou implementaci zvolen datový typ `brain float`. Vliv této redukce na výsledek se ukázal jako zanedbatelný. V případě použití typu `half` dopadly výsledky nepatrně hůře, implementace nad typem `brain float` proto byla zachována.

Jak zaznělo v návrhu, pro tyto koeficienty se ukázal prokládaný způsob ukládání dat jako ideální. Matice tak byly při výpočtech v redukované přesnosti sloučeny do jedné. Ušetřeny proto byly dvě jednotky místa, a to pro všechny implementované varianty snížené přesnosti.

Kappa koeficient

Kappa koeficient je reprezentován pouze jednou maticí s redukovanými dimenzemi. Opět zde platí, že obdobně jako u předchozích koeficientů zde nebylo jasné, který datový typ použít.

Z testů vyplynulo, že použití typu `brain float` přináší tak velké chyby, že je tento typ prakticky nepoužitelný. To platí především v kontrastu s tím, jak malé zlepšení složitosti přináší. V případě použití typu `half` se ukázala přesnost naopak dostačující. Nebylo přitom naraženo na problémy s možným rozsahem veličiny.

I přes použití typu `half` je nicméně vliv na přesnost výpočtů natolik signifikantní, že jsem se rozhodl tuto optimalizaci zařadit pouze pod rozšířenou redukovanou přesnost. V případě jejího užití se tak uživatel dočká další jedné jednotky uvolněného místa.

Kappa koeficient zdroje

Poslední redukovanou veličinou byl kappa koeficient zdroje. Stejně jako u kappa koeficientu se jedná pouze o jednu matici s redukovanými dimenzemi. Rozdíl je však v tom, že datový typ `half` byl jasně vhodnější, jelikož se do matice ukládají hodnoty funkce kosinus. Jedná se tak o poslední možnou jednotku uvolněného místa dle návrhu.

Veličina	Typ	Ušetřeno		
		Min	Mid	Max
Akustický koef. nelinearity	Half	2	2	2
Rychlost zvuku	Half	2	2	2
Hustota	Half	2	8	8
Podíly delty času a posunuté počáteční hustoty	Half	0	0	6
Zdroj počátečního tlaku	Brain float	2	2	2
Zdroj tlakových vln	Brain float	2	2	2
Absorpční koeficienty Tau a Eta	Brain float	4	4	4
Absorpční koeficienty Nabla	Brain float	2	2	2
Kappa koeficient	Half	0	0	1
Kappa koeficient zdroje	Half	1	1	1
Celkem ušetřeno		17	23	30

Tabulka 5.1: Souhrn provedených redukcí

Kapitola 6

Testování a zhodnocení dosažených výsledků

Tato kapitola bude věnována testování zhotovené implementace a vyhodnocení dosažených výsledků. Budou srovnány tři implementované úrovně výpočtů s redukovanou přesností. Testy byly provedeny na třech datasetech - `PH1-BM7-SC1`, `PH1-BM8-SC1` a `PH1-BM9-SC1`. Obsahem těchto datasetů je lidská lebka s mozkem. V přiložené tabulce 6.1 lze nalézt velikosti domén jednotlivých datasetů a počet kroků v jejich simulaci.

Při hodnocení bude nejprve důraz na jejich vliv na časovou úsporu a zrychlení konkrétních kernelů na základě provedených testů nad poskytnutými datasety. Dále bude věnována pozornost prostorový úsporám. V této sekci se budu snažit vyhodnotit teoretickou minimální i maximální ušetřenou paměť pomocí abstraktních jednotek paměti definovaných v sekci 5.4. Vyhodnoceny budou také naměřené úspory v reálném běhu aplikace. Poslední část bude věnována určení velikosti chyb výpočtů a srovnání relativní chyby mezi implementovanými třemi úrovněmi snížené přesnosti.

Dataset	Velikost dimenze			Počet kroků
	X	Y	Z	
PH1-BM7-SC1	324	192	192	3600
PH1-BM8-SC1	512	384	432	12000
PH1-BM9-SC1	512	512	432	12000

Tabulka 6.1: Velikost datasetů

6.1 Dosažené zrychlení

Nejprve byly provedeny souhrnné testy výkonu. Ty byly měřeny nad již výše zmíněnými třemi datasety. Program k-Wave při spuštění automaticky vypisuje informace o průběhu, očekávané době běhu a celkovém času výpočtů či běhu aplikace. Srovnávány budou fáze výpočtů aplikace na základě těchto údajů. Za zmínku také stojí, že konzistentně s očekáváními dle kapitoly 3 bylo při implementaci pozorováno zrychlení jak při fázi redukce dat do nižší přesnosti, tak při fázi úpravy výpočtů pro práci s dvojicemi hodnot.

Naměřené a vyhodnocené údaje komplexních testů lze nalézt v tabulkách 6.2, 6.3 a 6.4. Pro přehlednost zde budou údaje z testů s plnou přesností ve sloupci označeném `Full`

a údaje pro minimální, střední a maximální stupně redukované přesnosti ve sloupcích `Min`, `Mid` a `Max`.

Druhá polovina této podkapitoly se bude věnovat naměřeným datům z profilování jednotlivých kernelů za využití nástroje `nvprof`. V tabulce 6.7 lze nalézt dosažené rychlosti zápisu dat pro optimalizované kernely, které jsou využity simulacemi při testovaných datasetech. Z dat v této tabulce propustnosti paměti pro zápis bylo vycházeno při výpočtu celkového zrychlení konkrétních kernelů v následné tabulce 6.8.

Důležité je si uvědomit také, že zrychlení bylo v této práci docíleno primárně na zkrácení doby běhu kernelů. Větší část doby běhu je však závislá na výpočtu Fourierových transformací. Kromě vyhodnocení zkrácení celkové doby běhu programu je proto uvedeno také, jakého zrychlení bylo docíleno právě na kernelech. Tento údaj byl odvozen pomocí opakovaného měření, při kterém kernely nebyly spouštěny. Z toho byla odvozena jejich reálná doba běhu a docílené zrychlení.

	Full	Min	Mid	Max
Doba výpočtů [s]	20,09	19,39	18,79	18,51
Zrychlení výpočtů	0 %	3,61 %	6,92 %	8,54 %
Zrychlení kernelů	0 %	10,56 %	21,56 %	27,48 %

Tabulka 6.2: Doba běhu pro PH1-BM7-SC1

	Full	Min	Mid	Max
Doba výpočtů [s]	449,66	435,43	422,99	415,38
Zrychlení výpočtů	0 %	3,27 %	6,31 %	8,25 %
Zrychlení kernelů	0 %	8,99 %	18,30 %	24,81 %

Tabulka 6.3: Doba běhu pro PH1-BM8-SC1

	Full	Min	Mid	Max
Doba výpočtů [s]	596,77	577,75	560,94	550,42
Zrychlení výpočtů	0 %	3,33 %	6,39 %	8,42 %
Zrychlení kernelů	0 %	8,98 %	18,37 %	25,12 %

Tabulka 6.4: Doba běhu pro PH1-BM9-SC1

Z tabulek výše lze vypozorovat, že rozdíl mezi nízkými a středními redukcemi je pro dobu výpočtů prakticky stejně významný, jako rozdíl mezi minimálními redukcemi a plnou přesností výpočtů. To je zajímavé hlavně při uvedení do kontextu, že rozdíl mezi nízkou a střední redukovanou přesností je pouze v redukci matic hustoty. V případě hustoty přitom nebylo možné v některých případech zvolit přístup po dvojicích hodnot. To by bylo alespoň částečně umožněno, kdyby byly matice hustot ve směru `X` a ve směru `Y` ukládány prokládaně. To však nebylo v této práci implementováno kvůli nutnosti větších úprav již pro načítání dat. Lze se nicméně domnívat, že by tento přístup umožnil alespoň v kontextu těchto dvou matic dosáhnout vyššího zrychlení.

Profilování kernelů

Pro vyhodnocení zrychlení jednotlivých kernelů bylo také provedeno profilování za pomoci nástroje `nvprof`. Toto profilování bylo z časových důvodů provedeno pouze na nejmenším datasetu `PH1-BM7-SC1` při použití grafické karty Nvidia V100. Pro všechny z těchto datasetů se nicméně spouští totožné kernely, a tak by výsledky měly být obecně reprezentativní.

Ve výsledcích profilování došlo v poměrně velké části případů k mírnému snížení reálné propustnosti dat pro čteni, jak je vyobrazeno v tabulce 6.5. Obecně však při srovnání s následující tabulkou 6.6 lze vypozorovat v takových případech korelaci se snížením požadované rychlosti čtení z globální paměti. Takový výsledek je dle očekávání, jelikož na menší objem dat nyní připadá větší množství výpočetních operací a zápisů.

Kernel	Propustnost paměti - Čtení [GB/s]			
	Full	Min	Mid	Max
AddPressureSource	265,63	263,60	289,99	291,65
ComputeAbsorptionTerm	458,34	421,56	421,61	421,37
ComputeDensityLinear	534,81	530,69	579,36	577,76
ComputePressureGradient	227,67	227,76	227,98	196,88
ComputePressureTermsLinearPowerLaw	580,16	587,27	546,10	545,93
ComputeVelocityGradient	407,40	406,59	406,40	390,60
ComputeVelocityUniform	571,45	571,71	570,99	554,25
SumPressureTermsLinearPowerLaw	615,04	606,28	606,28	605,86

Tabulka 6.5: Propustnost paměti - čtení

Kernel	Žádaná rychlost Čtení [GB/s]			
	Full	Min	Mid	Max
AddPressureSource	407,19	380,44	462,08	465,27
ComputeAbsorptionTerm	458,33	421,56	421,61	421,36
ComputeDensityLinear	616,17	617,86	703,11	701,16
ComputePressureGradient	389,79	389,92	390,30	365,09
ComputePressureTermsLinearPowerLaw	580,16	587,27	546,09	545,93
ComputeVelocityGradient	531,72	530,68	530,42	518,97
ComputeVelocityUniform	639,05	639,35	638,55	633,13
SumPressureTermsLinearPowerLaw	615,03	606,27	606,27	605,85

Tabulka 6.6: Žádaná propustnost paměti - čtení

Naměřené rychlosti čtení však příliš nevypovídají o celkové rychlosti programu. Pro posouzení celkového zrychlení kernelů je naopak klíčová dosažená rychlost zápisu dat. Naměřené rychlosti zápisu lze nalézt v dále přiložené tabulce 6.7. Zde si lze všimnout, že pro některé kernely došlo k signifikantnímu propadu rychlosti zápisu, který není pouze statistickou chybou. Tento jev nastává z důvodu zápisu dat do matice s menším datovým typem. Pokud bychom tuto rychlost vyjádřili podle počtu zapsaných hodnot, byla by naopak signifikantně vyšší. V následující tabulce 6.8 jsou uvedeny zrychlení kernelů na základě dat z předešlé tabulky propustností a informací o datových typech zapisovaných dat.

Kernel	Propustnost paměti - Zápis [GB/s]			
	Full	Min	Mid	Max
AddPressureSource	148,34	157,18	150,10	150,77
ComputeAbsorptionTerm	306,43	337,70	337,45	337,51
ComputeDensityLinear	228,39	245,15	175,15	174,71
ComputePressureGradient	455,06	455,41	456,00	472,64
ComputePressureTermsLinearPowerLaw	166,80	181,90	219,39	219,36
ComputeVelocityGradient	348,15	348,87	348,91	360,22
ComputeVelocityUniform	190,79	191,33	190,79	221,63
SumPressureTermsLinearPowerLaw	107,90	138,66	138,61	138,62

Tabulka 6.7: Propustnost paměti - zápis

V tabulce 6.8 lze vidět procentuální zrychlení jednotlivých kernelů. Důležité je podotknout, že zrychlení pod hodnotou 0,3 % jsou všechna zaznamenána v kernelech, ve kterých pro danou úroveň optimalizací k žádným optimalizacím nedošlo. Jedná se tedy buď o chybu měření, nebo vedlejší efekt optimalizací v jiných částech programu, díky kterým mohlo dojít k drobnému uvolnění přenosového pásma.

Kernel	Zrychlení kernelu		
	Min	Mid	Max
AddPressureSource	5,96 %	102,37 %	103,28 %
ComputeAbsorptionTerm	10,20 %	10,12 %	10,14 %
ComputeDensityLinear	7,34 %	53,38 %	52,99 %
ComputePressureGradient	0,08 %	0,21 %	3,86 %
ComputePressureTermsLinearPowerLaw	9,05 %	31,53 %	31,51 %
ComputeVelocityGradient	0,21 %	0,22 %	3,47 %
ComputeVelocityUniform	0,28 %	0,00 %	16,16 %
SumPressureTermsLinearPowerLaw	28,51 %	28,46 %	28,47 %

Tabulka 6.8: Zrychlení kernelů

V případě prvního kernelu v tabulce 6.8 výše je dále zajímavé, že došlo ke zrychlení o více než 100 %, a to ačkoliv v tomto případě nešlo provést optimalizaci korektně z pohledu přístupu po dvojicích hodnot. Ten v tomto případě nelze aplikovat, jelikož pro tento kernel nejsou všechny přístupy sekvenční, nýbrž dochází k přístupům na základě matice indexů. Přesto v případě tohoto kernelu došlo ke zvýšení datové propustnosti pro čtení i zápis. Důvodem je pravděpodobně, že nesekvenčnímu přístupu více napomáhá paměť cache a zmenšení objemu dat vedlo k jejímu lepšímu využití. Tuto tezi potvrzují i data z profileru, kde četnost přístupů do paměti L1 cache pro tento kernel stoupla z průměrných 56,69 % na 57,37 % pro minimální optimalizace, na 61,03 % pro střední optimalizace a na 61,21 % pro maximální optimalizace. Všechny naměřené hodnoty četnosti přístupů do L1 cache lze nalézt v přiložené tabulce 6.9. Obdobné zvýšení četnosti přístupů do L1 cache lze pozorovat i v případě ostatních kernelů.

Pro ostatní kernely je výsledné zrychlení víceméně úměrné snížení nároků na využitý objem dat. Jejich zrychlení tak odpovídá očekáváním.

Kernel	L1 cache - Hit Rate			
	Full	Min	Mid	Max
AddPressureSource	56,69%	57,37%	61,03%	61,21%
ComputeAbsorptionTerm	30,08%	34,22%	35,21%	33,49%
ComputeDensityLinear	35,75%	37,66%	41,69%	42,12%
ComputeVelocityUniform	29,81%	30,16%	30,01%	32,92%
ComputePressureGradient	36,14%	36,04%	36,18%	37,68%
ComputePressureTermsLinearPowerLaw	0,00%	0,00%	0,00%	0,00%
ComputeVelocityGradient	47,06%	47,10%	47,25%	48,91%
SumPressureTermsLinearPowerLaw	0,00%	0,00%	0,00%	0,00%

Tabulka 6.9: Četnost přístupů do cache L1

V tabulce 6.10 lze vidět celkovou docílenou propustnost globální paměti. Pozitivní efekt na celkovou propustnost má zřejmě přístup po dvojicích. K drobnému zpomalení pro vyšší míry optimalizace naopak obecně dochází, protože načítání po 4 bytech (pro `half2`) je méně efektivní, než načítání po 8 bytech (pro `float2`), jak bylo ukázáno v grafu 3.3. Teoretický limit propustnosti paměti pro kartu Nvidia V100 je 900 GB/s [15]. I při jednoduchém testu (viz graf 3.3), se však nepodařilo přesáhnout rychlost 400 GB/s kopírování. Realistická rychlost čtení + zápisu je tak pod 800 GB/s, naměřené hodnoty jsou tedy blízko limitu.

Kernel	Propustnost paměti - Celkem [GB/s]			
	Full	Min	Mid	Max
AddPressureSource	413,97	420,78	440,09	442,42
ComputeAbsorptionTerm	764,77	759,26	759,06	758,88
ComputeDensityLinear	763,20	775,84	754,51	752,47
ComputePressureGradient	682,73	683,17	683,98	669,52
ComputePressureTermsLinearPowerLaw	746,96	769,17	765,49	765,29
ComputeVelocityGradient	755,55	755,46	755,31	750,82
ComputeVelocityUniform	762,24	763,04	761,78	775,88
SumPressureTermsLinearPowerLaw	722,94	744,94	744,89	744,48

Tabulka 6.10: Dosažená propustnost paměti

6.2 Paměťová úspora

Úspora v množství potřebné paměti pro běh programu k-Wave je druhým důležitým pozitivem této práce. V kapitole 5.4 je podrobně uvedené, jakých redukcí bylo docíleno. V této části práce bude uvedena nejprve naměřená paměťová úspora na dodaných datasetech. Poté bude analyzována teoretická minimální a maximální úspora na základě údajů shrnutých v tabulce 5.1 na konci předešlé kapitoly.

V reálných experimentech byly testy provedeny nejprve na aktuální implementaci, ve které se vstup zdroje tlakových vln ukládá v paměti najednou pro všechny časové kroky. Poté byly provedeny testy, při kterých byla paměť omezena na standardní velikost jedné matice s plnými dimenzemi. Tím bylo simulováno bufferování dat pro každý časový krok zvlášť, což by měl být v blízké době nasazený update. Díky tomu byl určen reálný efekt v této prozatím nenasazené verzi programu k-Wave. V tabulkách 6.11, 6.12 a 6.13 lze nalézt naměřené paměťové údaje a zpracované informace o procentuální redukci potřebné paměti.

	Full	Min	Mid	Max
Paměť CPU [MB]	9870	13959	13890	13813
Paměť GPU [MB]	10149	5825	5759	5681
Paměť CPU (bufferovaný zdroj) [MB]	1486	1395	1327	1246
Paměť GPU (bufferovaný zdroj) [MB]	1769	1635	1569	1491
Ušetřeno CPU paměti (bufferovaný zdroj)	0 %	6,12 %	10,7 %	16,15 %
Ušetřeno GPU paměti (bufferovaný zdroj)	0 %	7,57 %	11,31 %	15,72 %

Tabulka 6.11: Využití paměti pro dataset PH1-BM7-SC1

	Full	Min	Mid	Max
Paměť CPU [MB]	36368	49575	49089	48519
Paměť GPU [MB]	37203	22347	21859	21289
Paměť CPU (bufferovaný zdroj) [MB]	8595	7943	7459	6890
Paměť GPU (bufferovaný zdroj) [MB]	9441	8465	7977	7409
Ušetřeno CPU paměti (bufferovaný zdroj)	0 %	7,59 %	13,22 %	19,84 %
Ušetřeno GPU paměti (bufferovaný zdroj)	0 %	10,34 %	15,51 %	21,52 %

Tabulka 6.12: Využití paměti pro dataset PH1-BM8-SC1

	Full	Min	Mid	Max
Paměť CPU [MB]	39016	51953	51305	50548
Paměť GPU [MB]	40071	24943	24293	23535
Paměť CPU (bufferovaný zdroj) [MB]	11350	10483	9837	9079
Paměť GPU (bufferovaný zdroj) [MB]	12417	11115	10467	9709
Ušetřeno CPU paměti (bufferovaný zdroj)	0 %	7,64 %	13,33 %	20,01 %
Ušetřeno GPU paměti (bufferovaný zdroj)	0 %	10,49 %	15,70 %	21,81 %

Tabulka 6.13: Využití paměti pro dataset PH1-BM9-SC1

V tabulkách výše si lze povšimnout, že při spuštění s redukovanou přesností je v aktuální verzi zapotřebí vyšší množství paměti RAM pro CPU. To je způsobeno tím, že velká data pro zdroj tlakových vln mají dočasně alokováno 1,5 krát větší množství paměti ve srovnání s verzí s plnou přesností, a to kvůli načtení dat ze souboru před jejich následnou redukcí. Tento přístup ovšem není problémem, jelikož paměť pro CPU není ve srovnání s pamětí GPU tolik omezená. V budoucí verzi programu k-Wave navíc nemá být tato jedna matice tak signifikantní díky postupnému načítání dat, a tak se tento problém úplně ztratí.

Dále lze obecně sledovat trend zvyšujícího se procenta ušetřené paměti pro větší simulace. Tento jev je pravděpodobně způsoben primárně snižující se poměrovou částí nutné paměti pro jednorozměrná a dvourozměrná data vůči redukovaným třírozměrným datům.

V teoretické rovině lze vyčíslit odhadovaný minimální a maximální podíl ušetřené paměti na základě tabulky 5.1. Stačí pro to spočítat, kolik jednotek místa je zapotřebí přinejmenším (tj. s minimálním možným množstvím matic) a kolik maximálně. To lze vidět v následující tabulce 6.14 pro všechny úrovně výpočtů s redukovanou přesností. Zde je pro nejhorší případ počítáno, že žádná z volitelných matic, která by mohla být redukována, není použita. Pro minimální úroveň redukcí to tak může znamenat i nulovou redukci. Pro nejlepší redukci

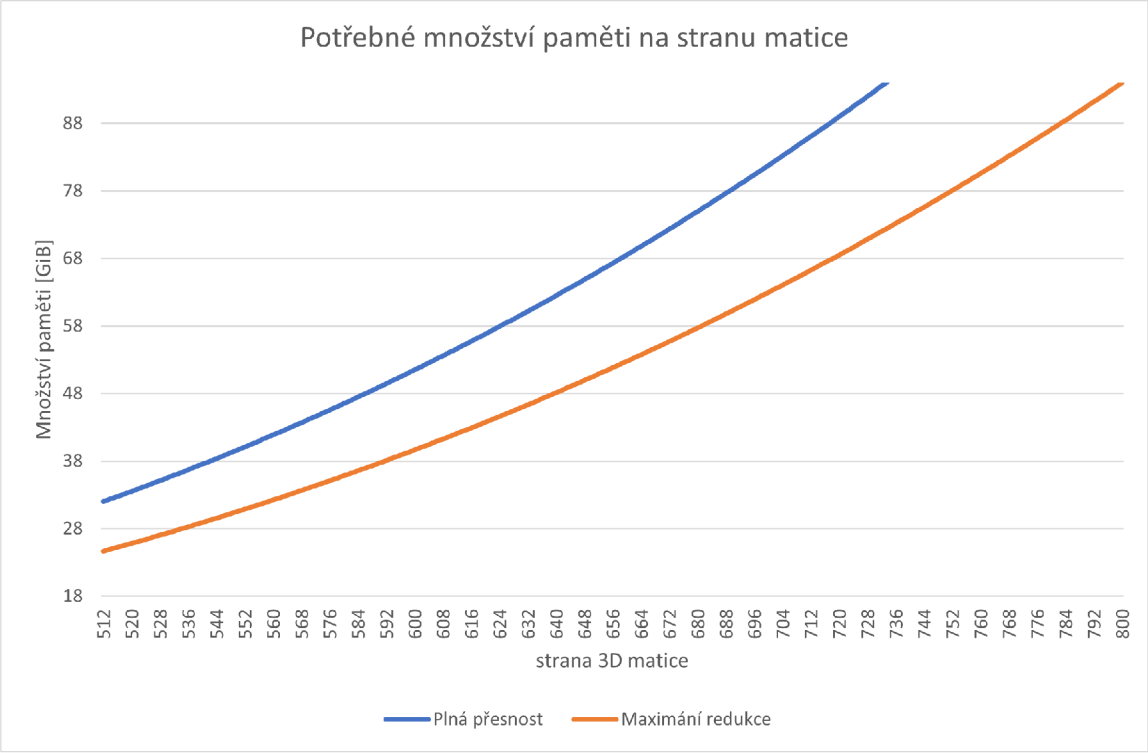
je naopak počítáno s případem, kdy by byly použity všechny matice, které byly pro daný stupeň optimalizace převedeny do nižší přesnosti.

	Min	Mid	Max
Redukováno/potřebováno (nejhorší)	0/94	6/94	7/82
Redukce paměti (minimálně)	0 %	6,38 %	8,54 %
Redukováno/potřebováno (nejlepší)	17/116	23/116	30/128
Redukce paměti (maximálně)	14,66 %	19,83 %	23,44 %

Tabulka 6.14: Limity ušetřené paměti

Nyní lze porovnat teoretické limity v tabulce výše s dříve naměřenými hodnotami. Lze si všimnout, že obzvlášť maximální redukce pro maximální optimalizace je velmi blízko s údaji v posledním provedeném testu, které lze vidět v tabulce 6.13. To platí, ačkoliv minimálně některé parametry, které mohou podstoupit redukci, nebyly použity. Lze tedy předpokládat, že se zvětšujícími se maticemi dat by se procento ušetřené paměti již příliš nezvedalo. Naopak to také potvrzuje přibližnou správnost spočtených teoretických údajů.

Následující graf 6.1 ukazuje přibližnou teoretickou potřebu paměti při využití všech veličin pro plnou přesnost výpočtů a pro maximální implementovanou redukci přesnosti s reálnou redukcí potřebné paměti přibližně 23 %. Specificky vyobrazuje využití do velikosti cca 94 GiB paměti, což je množství paměti na specializované GPU Nvidia H100 NVL [28]. Ač by na takovém GPU šlo provozovat program k-Wave s velikostí strany redukovaných vstupních matic přibližně 800, pro výpočty v plné přesnosti by to bylo pouze zhruba 717.



Obrázek 6.1: Porovnání využití paměti

6.3 Přesnost výsledků

Tato sekce bude věnována dosaženým výsledkům testů přesnosti simulací nad testovanými datasety. Pro potřeby srovnání algoritmu s plnou přesností vůči variantám s redukovanou přesností budou spočítány L^2 a L^∞ chyby, které jsou relativní vůči ohnisku tlaku (tj. bodu nejvyššího tlaku). Dále bude uveden relativní rozdíl těchto ohnisek a jejich vzdáleností. Vzorce pro výpočet těchto metrik lze nalézt v tabulce 6.15 níže. Hodnoty p_1 a p_2 značí hodnoty tlaku pro plnou a redukovanou přesnost. Hodnota $max(p_x)$ pak označuje hodnotu v ohnisku dané matice a $posmax(p_x)$ pozici ohniska.

Metrika	Definice
L^2 chyba	$\sqrt{\frac{\sum(p_1-p_2)}{\sum(p_1{}^2)}}$
L^∞ chyba	$\frac{max\lvert p_1-p_2\rvert}{max(p_1)}$
Rozdíl amplitud ohnisek	$\frac{\lvert max(p_1)-max(p_2)\rvert}{max(p_1)}$
Vzdálenost ohnisek	$\lvert\lvert pos\ max(p_1) - pos\ max(p_2)\rvert\rvert_2$

Tabulka 6.15: Definice metrik

Kromě výše uvedených metrik budou poskytnuty grafy vyobrazující vrstvu s ohniskem tlaku a grafy chyby v této vrstvě. Grafické reprezentace budou vždy vyobrazovat řezy skrze osy X-Y (vlevo) a X-Z (vpravo) v lokaci ohniska tlaku. Dále pak budou pro jednotlivé datasety vyobrazeny relativní chyby vůči ohnisku pomocí krabicových grafů.

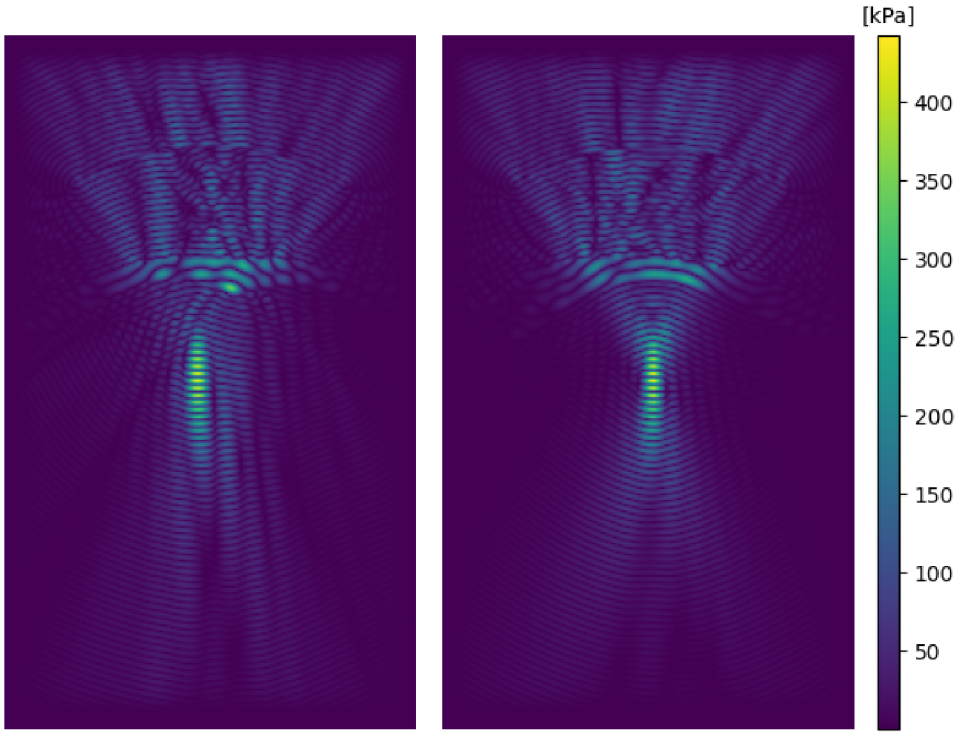
Dataset H1-BM7-SC1

V tabulce 6.16 lze vidět velmi pozitivní data týkající se ohnisek. Prvním pozitivním faktem je, že pozice zůstala nezměněná. Dalším pozitivem je fakt, že rozdíly v tlaku ohniska se napříč testy téměř neliší. V případě minimální verze optimalizací s největší mírou přesnosti se všechny tři údaje chyby navzájem téměř neliší. V kontextu s L^2 a L^∞ chybami pro větší optimalizace lze konstatovat, že největší chyby vznikají spíše v jiných částech matice výsledných hodnot tlaku.

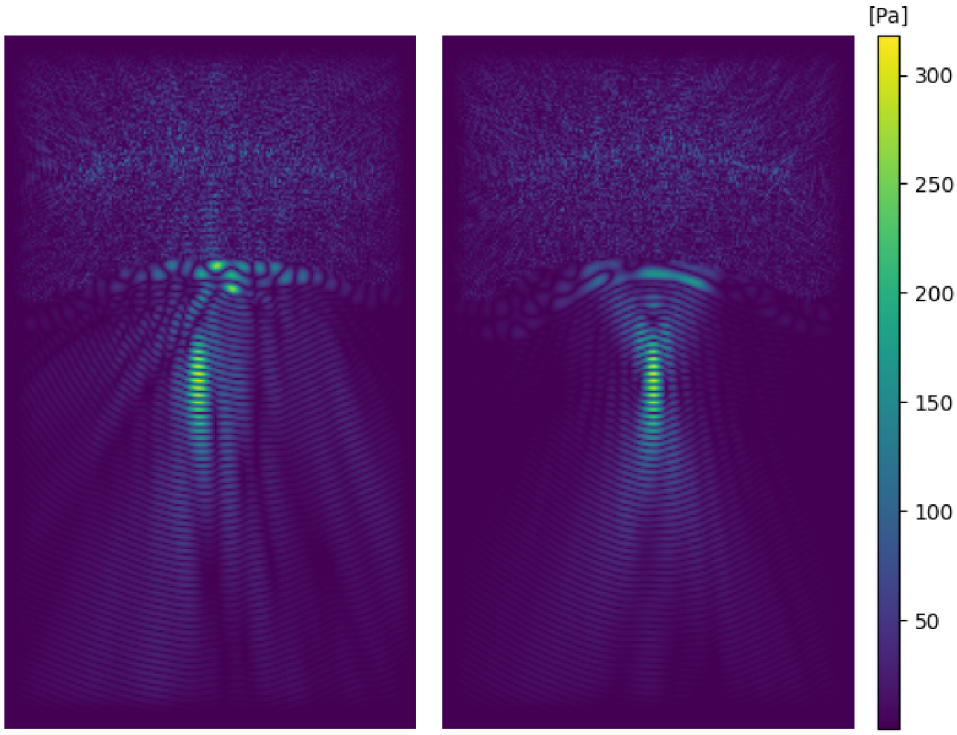
Metrika	Min	Mid	Max
L^2 chyba	0,071 %	0,415 %	1,166 %
L^∞ chyba	0,081 %	0,313 %	1,004 %
Rozdíl amplitud ohnisek	0,072 %	0,045 %	0,065 %
Vzdálenost ohnisek	0	0	0

Tabulka 6.16: Metriky pro H1-BM7-SC1

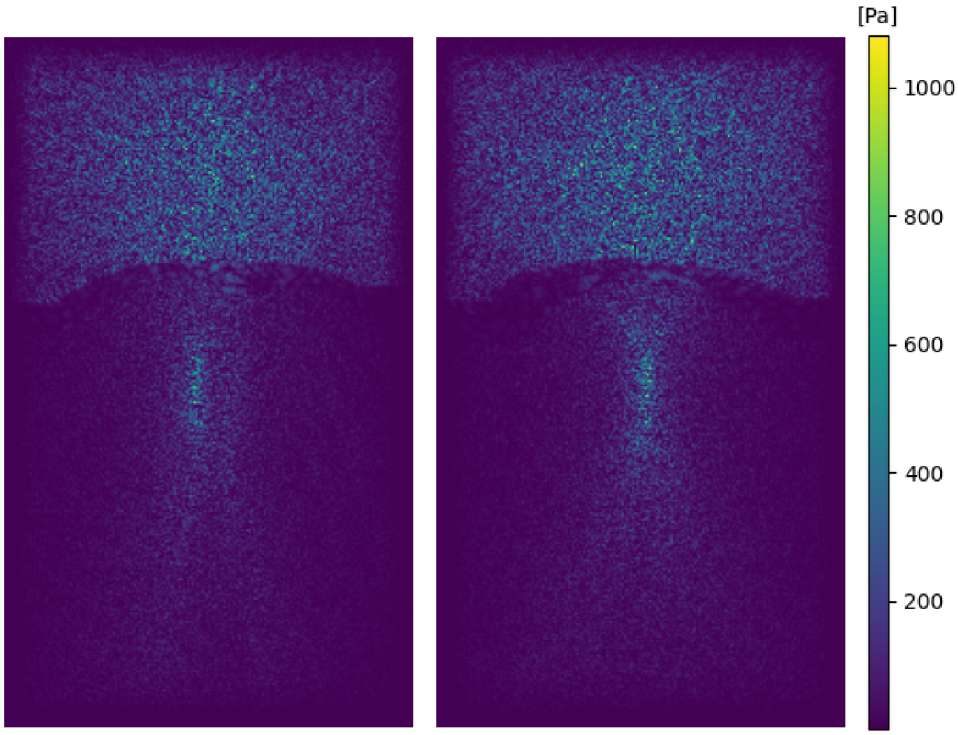
Na obrázku 6.2 lze vidět výslednou matici tlaku pro řezy skrz ohnisko tlaku. Vzhledem k velikosti chyb nelze pozorovat rozdíly v grafech této vrstvy pro různé přesnosti, a tak nejsou varianty grafu pro redukované přesnosti zobrazeny. Místo toho jsou níže uvedeny grafy 6.3, 6.4 a 6.5, které reprezentují chybu výpočtů v této vrstvě.



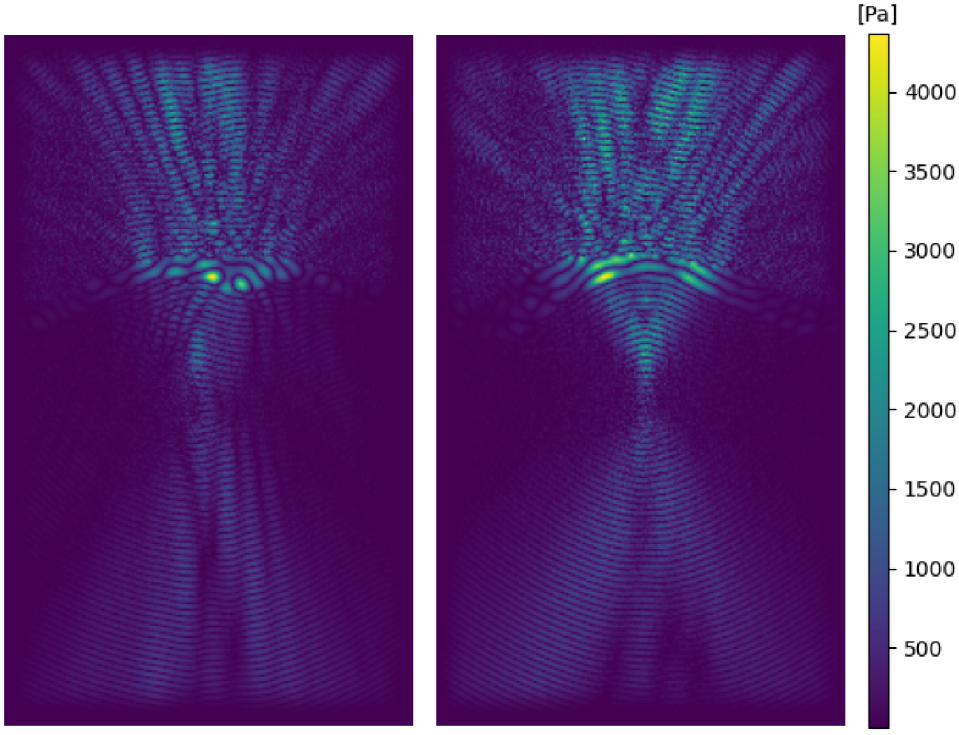
Obrázek 6.2: Vrstva s ohniskem tlaku - PH1-BM7-SC1



Obrázek 6.3: Odchylka vrstvy s ohniskem - Minimální redukce - PH1-BM7-SC1



Obrázek 6.4: Odchylka vrstvy s ohniskem - Střední redukce - PH1-BM7-SC1



Obrázek 6.5: Odchylka vrstvy s ohniskem - Maximální redukce - PH1-BM7-SC1

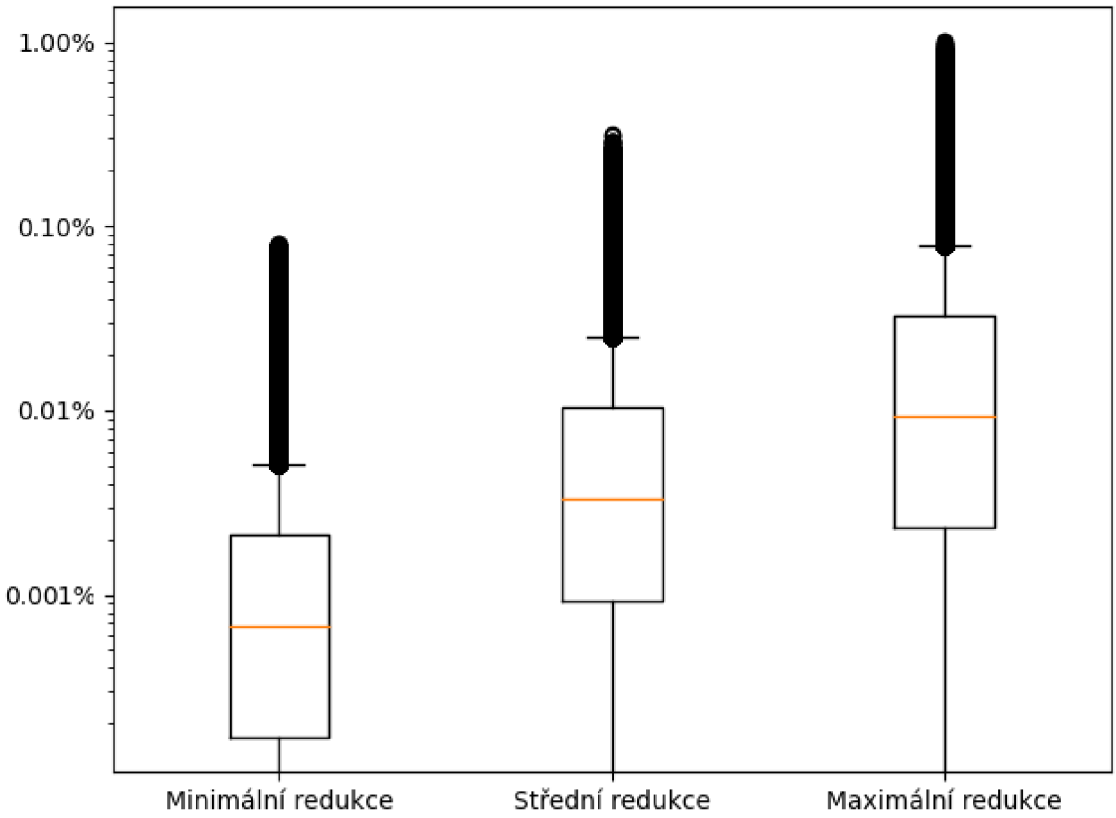
V případě grafu 6.3, který reprezentuje výpočty s minimálními redukcemi, lze vidět, že největší chyby jsou relativně konzistentní vůči výši hodnot tlaku. Toto chování se víceméně dalo očekávat z relativně konzistentní chyby ohniska i L^2 metriky.

Pro graf 6.4 se středními optimalizacemi jsou data zřetelně odlišná. V tomto případě jsou chyby mnohem více zašuměné. Důvodem výrazně větší chyby L^2 je pak pravděpodobně šum v oblasti, odkud se šíří tlakové vlny před stykem s objektem uprostřed. Nicméně i v okolí ohniska se vyskytují body se zřetelně vyšší chybou.

Graf 6.5 znázorňuje odchylky pro nejvyšší míru optimalizací. Zde lze vidět, že největší chyby zřejmě vznikají v místě lebeční kosti. Mimo to opět platí, že došlo k výraznému nárůstu chyby v místě, odkud se šíří akustické vlny. Pro samotné ohnisko však dochází téměř k vynulování výsledné chyby.

Celkově lze konstatovat, že každá z optimalizací zřejmě přináší chybu do odlišných částí výsledků. Ohnisko, však obecně vzato není optimalizacemi příliš dotčeno.

Obrázek 6.6 vyobrazuje krabicové grafy normalizovaných odchylek vůči ohnisku. Lze zde vidět, že typické chyby jsou ve všech případech alespoň desetkrát menší, než je nejvyšší odlehlá hodnota. I v případě maximální redukce jsou tak typické chyby menší než 0,08 % a pro minimální redukce je po zanedbání odlehlých hodnot maximální chyba dokonce jen zhruba 0,005 %.



Obrázek 6.6: Grafy normalizovaných odchylek vůči ohnisku - PH1-BM7-SC1

Dataset H1-BM8-SC1

V případě tohoto datasetu je opět pozitivní, že ohnisko zůstalo na stejném místě. Ve srovnání s předchozím datasetem `H1-BM7-SC1`, který je oříznutou verzí tohoto datasetu, však lze v tabulce 6.17 zřetelně vidět nárůst chyb. Dokonce ani chyba ohniska tentokrát není tak stabilní, ačkoliv je stále nižší než další sledované metriky. V případě minimálních redukcí a relativní chyby L^∞ jsou však hodnoty porovnatelné s předchozím testem.

Metrika	Min	Mid	Max
L^2 chyba	0,091 %	0,519 %	2,765 %
L^∞ chyba	0,080 %	0,301 %	1,014 %
Rozdíl amplitud ohnisek	0,071 %	0,197 %	0,264 %
Vzdálenost ohnisek	0	0	0

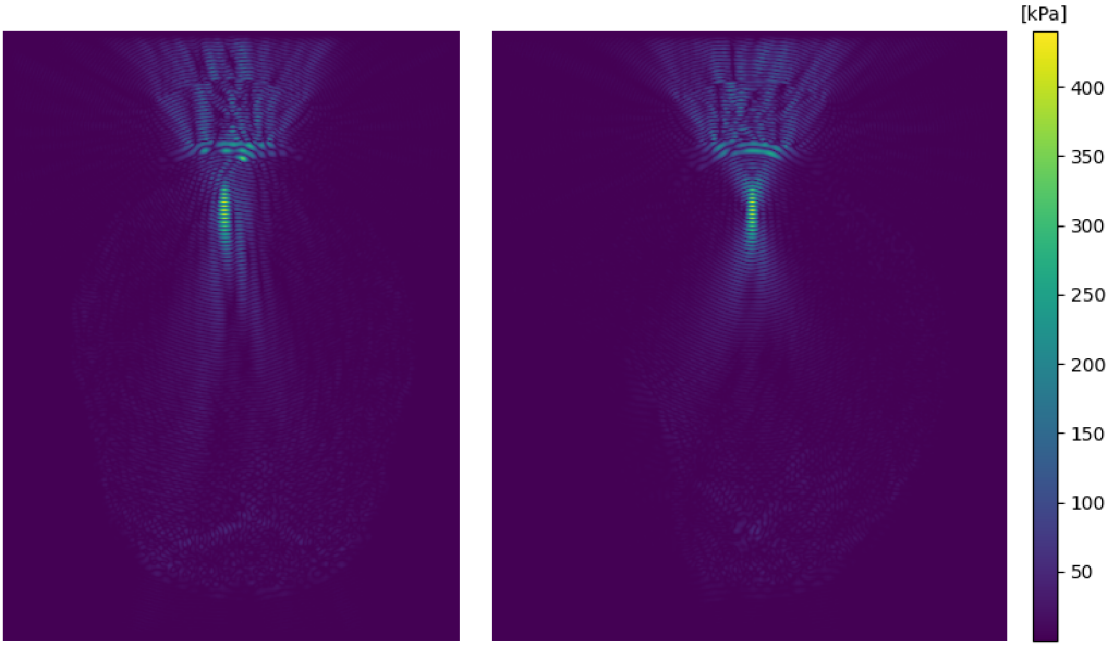
Tabulka 6.17: Metriky pro H1-BM8-SC1

Na grafech v obrázku 6.7 je opět vidět řezy ohniskem skrze osy X-Y a X-Z. Znovu zde platí, že v grafické reprezentaci řezu nejsou viditelné rozdíly mezi plnou přesností a výsledky v redukované přesnosti. Obrázky 6.8, 6.9 a 6.10 proto znovu vyobrazují samotnou chybu. Lze přitom sledovat podobné chování, jaké bylo vidět u testování přesnosti při předchozím datasetu.

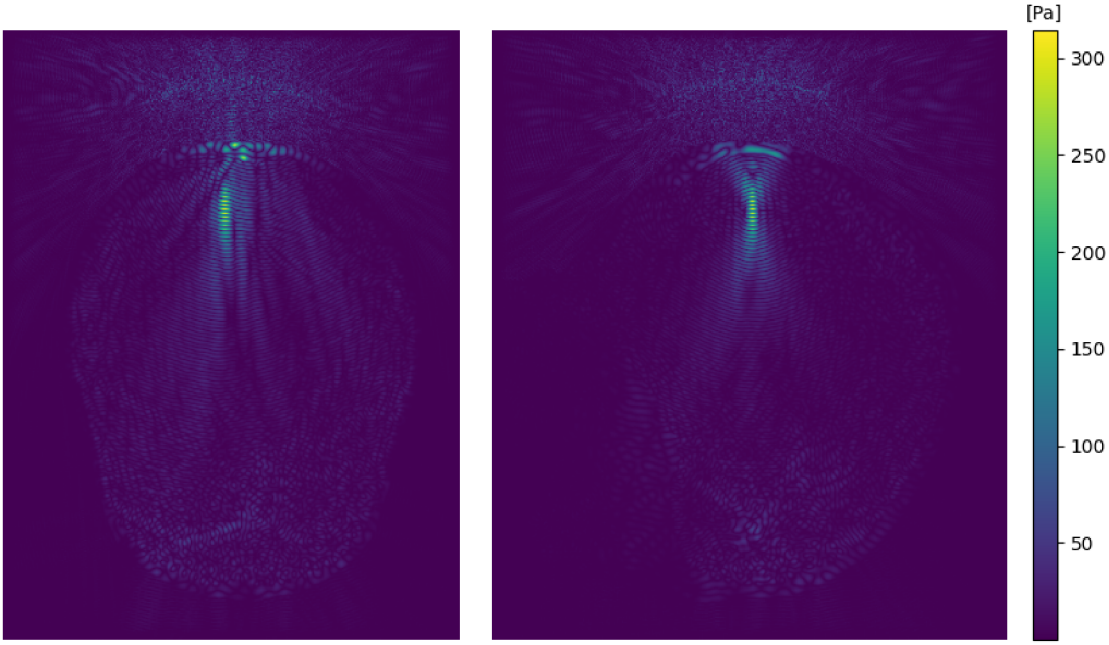
Řezy v obrázku 6.8 vyobrazují chybu při minimální redukci. Stejně jako v předešlém testu u obrázku 6.3 je výše chyby viditelně korelující s výší tlaku.

V případě grafů na obrázku 6.9 lze pozorovat chybu při střední úrovni optimalizací. I zde se opakuje vzor z 6.4, kdy chyba začíná být více zašuměná a nachází se více v oblasti, odkud se šíří akustické vlny, než v samotném objektu, ve kterém se nachází ohnisko tlaku.

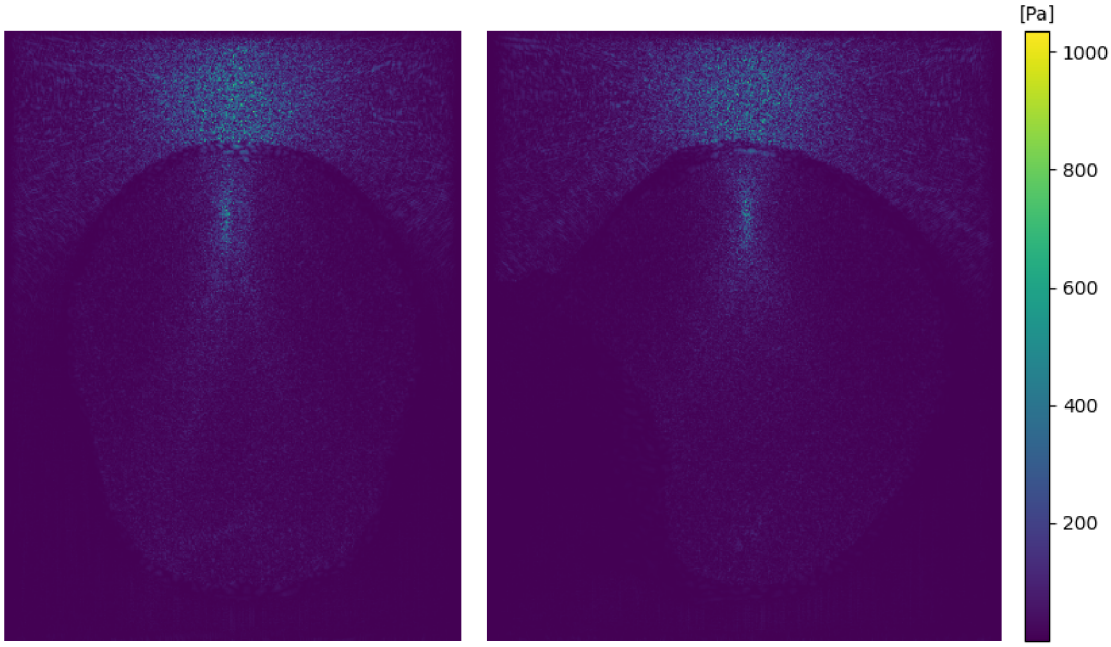
Zobrazení chyb v obrázku 6.9 podává ve srovnání s obrázkem 6.5 největší viditelné rozdíly. Rozdíl byl očekávatelný již dle razantně větší L^2 chyby a také vyšší chyby ohniska. Obdobné je chování, co se týče velikosti chyb v oblasti, odkud se šíří akustické vlny. Zásadně větší chyba je však vidět skrz celou strukturu mozku, což je nejspíše hlavní důvod zvýšené L^2 chyby. Důležité je si nicméně uvědomit, že odchylky se stále pohybují pod výší 1 % z ohniskové hodnoty tlaku.



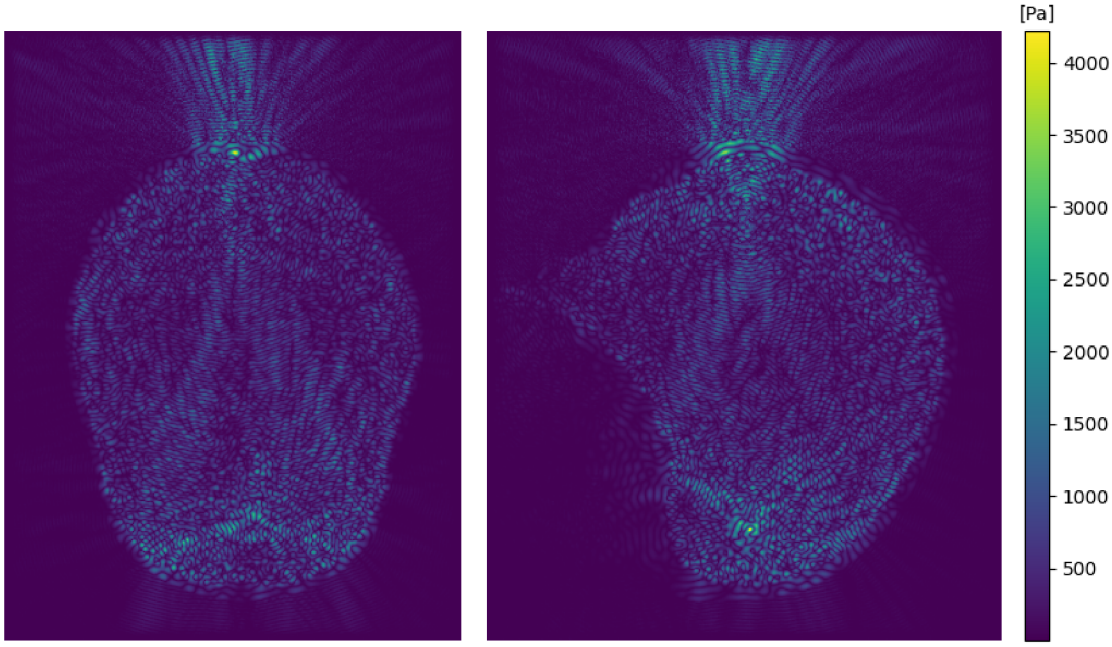
Obrázek 6.7: Vrstva s ohniskem tlaku - PH1-BM8-SC1



Obrázek 6.8: Odchylka vrstvy s ohniskem - Minimální redukce - PH1-BM8-SC1

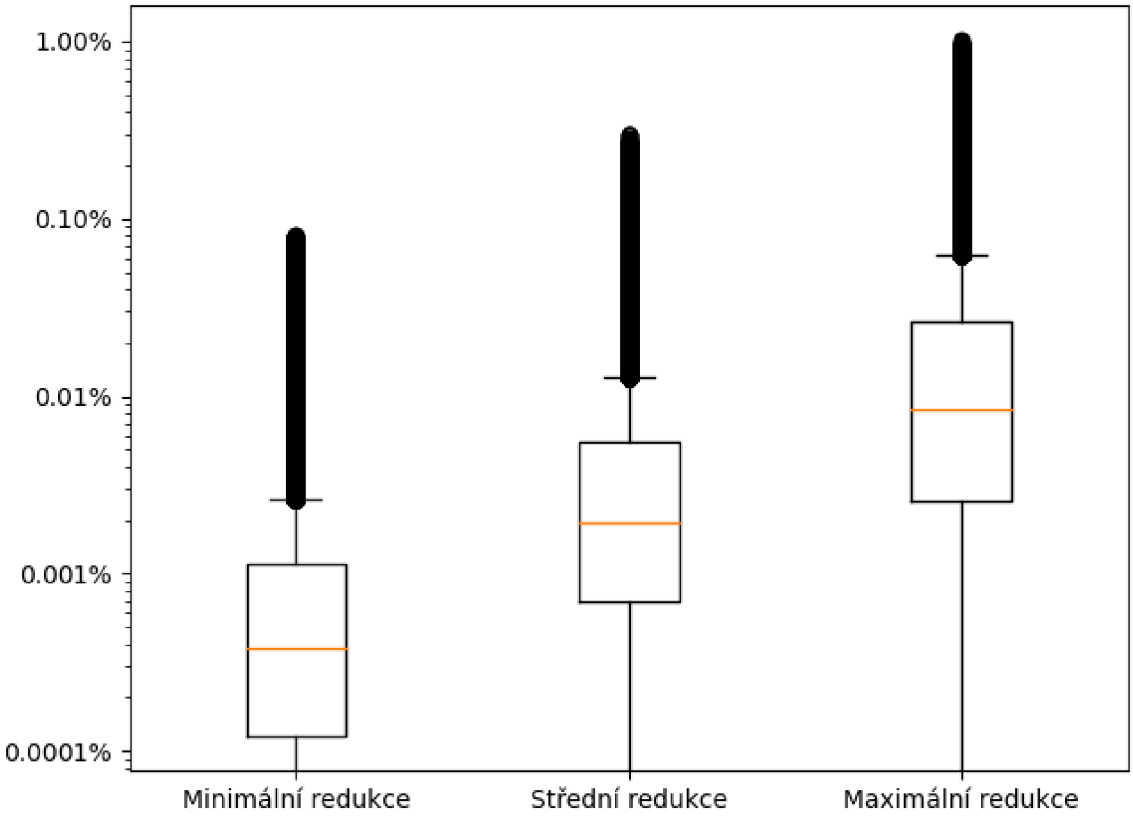


Obrázek 6.9: Odchylka vrstvy s ohniskem - Střední redukce - PH1-BM8-SC1



Obrázek 6.10: Odchylka vrstvy s ohniskem - Maximální redukce - PH1-BM8-SC1

Krabicové grafy znázorněné v obrázku 6.11 jsou téměř identické, jako byly krabicové grafy znázorněné v obrázku 6.6 u předchozího datasetu. Patrně největším rozdílem je obecně vyšší rozsah odlehlých hodnot. Odlehlé hodnoty pro minimální redukci dosahují stejně jako u minulého datasetu výše cca 0,1 %. Po jejich zanedbání však typická chyba dosahuje maximálně pouze zhruba 0,003 %. Obdobné chování lze vidět i pro grafy střední a maximální redukce.



Obrázek 6.11: Grafy normalizovaných odchylek vůči ohnisku - PH1-BM8-SC1

Dataset H1-BM9-SC1

V tabulce 6.18 níže lze vidět sledované veličiny pro dataset `H1-BM9-SC1`. Stejně jako u předchozích testů zůstává velkým pozitivem, že ohnisko zůstalo na zcela totožném místě. V případě minimální redukce jsou data srovnatelná s daty minulého datasetu v tabulce 6.17, pouze došlo k malému nárůstu L^∞ chyby. Pro střední redukce došlo k nárůstu L^2 a L^∞ chyby, rozdíl amplitud ohnisek je však naopak nižší. Pro maximální redukce je nárůst chyb pro tento větší dataset nejzřetelnější, L^2 chyba vzrostla zhruba 2,5násobně, L^∞ chyba pak vzrostla na téměř trojnásobné procento. To lze z části přisoudit nižší ohniskové hodnotě tlaku v tomto testu, vůči kterému jsou chyby vztaženy. Celkově však lze předpokládat, že je vyšší chyba zaviněna primárně postupnou akumulací chyb v důsledku větší simulace.

Metrika	Min	Mid	Max
L^2 chyba	0,092 %	0,731 %	6,980 %
L^∞ chyba	0,106 %	0,542 %	2,941 %
Rozdíl amplitud ohnisek	0,070 %	0,132 %	0,530 %
Vzdálenost ohnisek	0	0	0

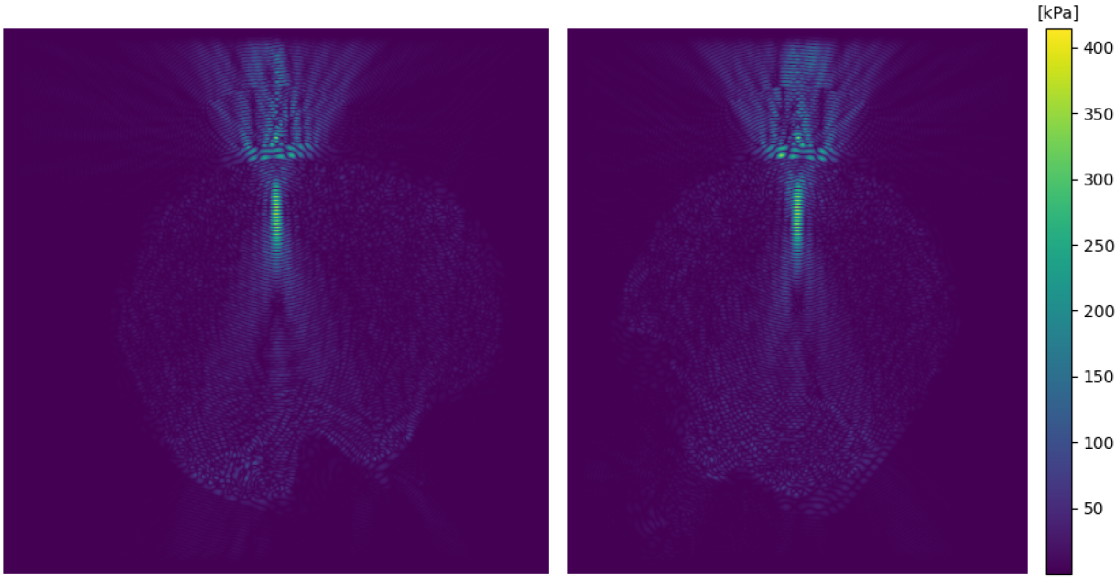
Tabulka 6.18: Metriky pro H1-BM9-SC1

I přes celkový nárůst chyb nelze v grafické reprezentaci řezů skrze ohnisko vidět rozdíl. Stejně jako u předchozích testů je tak na obrázku 6.12 vykreslen tlak v řezech při výpočtech s plnou přesností. Grafy v obrázcích 6.13, 6.14 a 6.15 pak zobrazují výši chyb v těchto řezech.

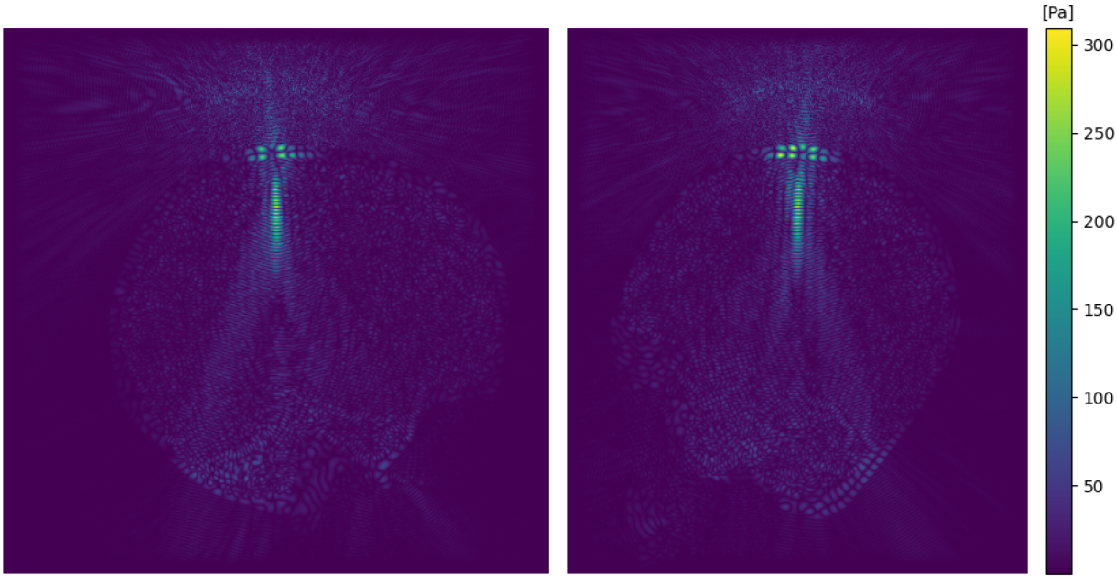
V obrázku 6.13 lze vidět chybu při minimální redukci. Stejně jako u obrázků 6.3 a 6.8 z předchozích testů je viditelná korelace chyby s hodnotou tlaku, přičemž nejvyšší chyba je v místě ohniska a u lebeční kosti v místě, odkud se šíří akustické vlny. Výše chyby je v absolutních číslech pro všechny 3 testy téměř totožná. Lze tedy očekávat podobné výsledky i v případě dalších testů.

Na obrázku 6.14 je vyobrazena chyba pro střední redukci přesnosti. Chyba je znovu ve srovnání s minimální redukcí mnohem více zašuměná. Pro celou oblast mozku se v tomto případě zdá chyba téměř totožná, nicméně pouze jako šum. Největší chyba se znovu ukazuje hlavně v místě, odkud se šíří akustické vlny, což není pro potřeby této simulace významným ukazatelem. Drobný nárůst lze nicméně pozorovat i v místě ohniska.

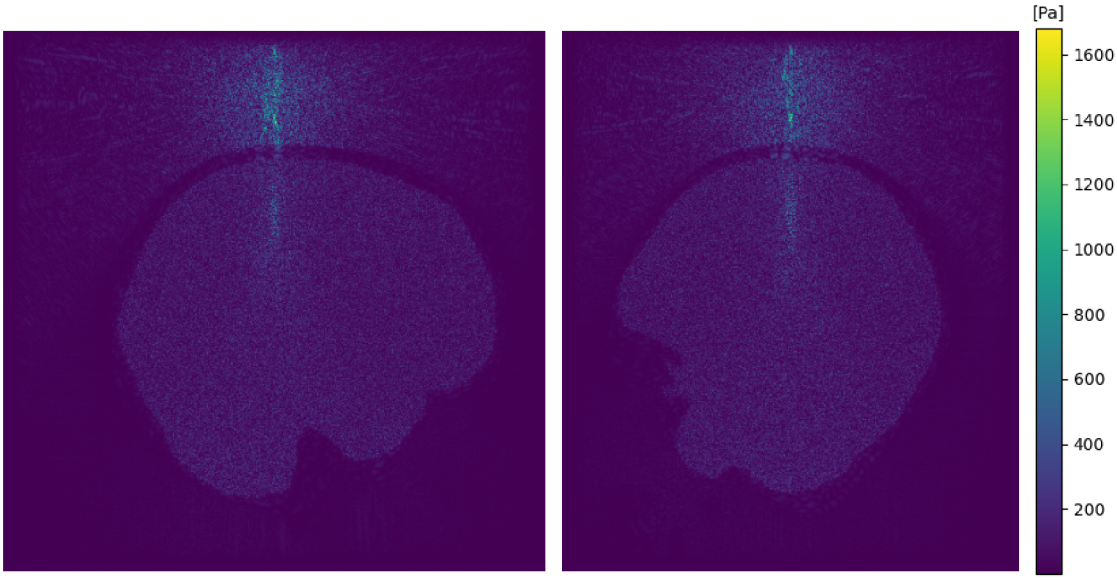
Celkově největší naměřené chyby napříč všemi testy lze pozorovat na řezech v obrázku 6.15. Poměrně velké chyby se tentokrát projevují napříč celým mozkem. Ty jsou v tomto případě v absolutních číslech 2,5krát vyšší, než byly při minulém testu v obrázku 6.10. Narozdíl od obrázků 6.5 a 6.10 lze navíc i při tomto zobrazení chyby tentokrát zřetelně vidět oblast ohniska tlaku. Oblast, odkud se šíří akustické vlny, se naopak setkává s relativně zanedbatelnějšími chybami.



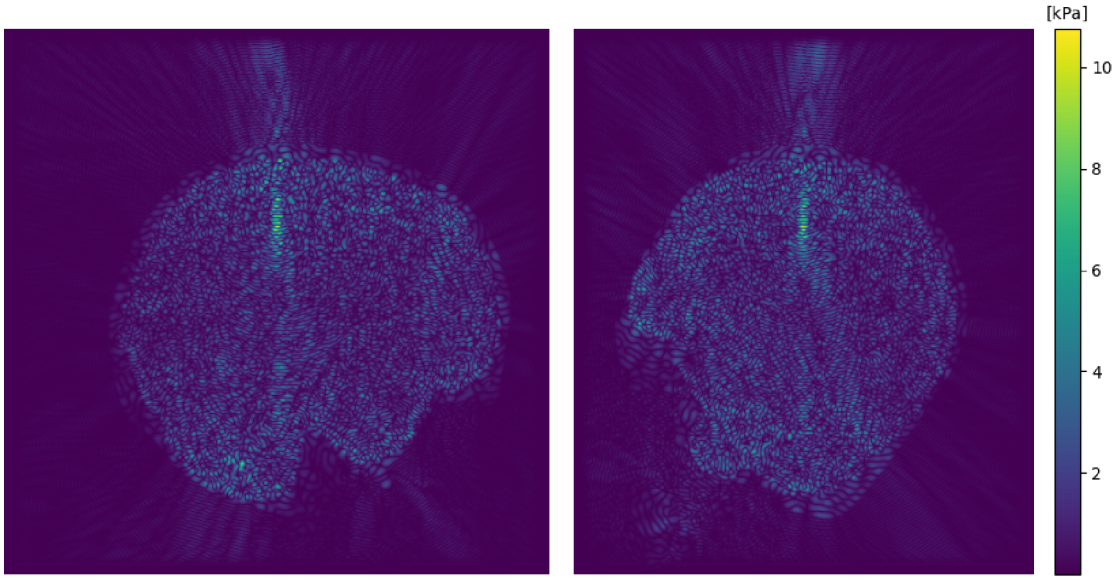
Obrázek 6.12: Vrstva s ohniskem tlaku - PH1-BM9-SC1



Obrázek 6.13: Odchylka vrstvy s ohniskem - Minimální redukce - PH1-BM9-SC1

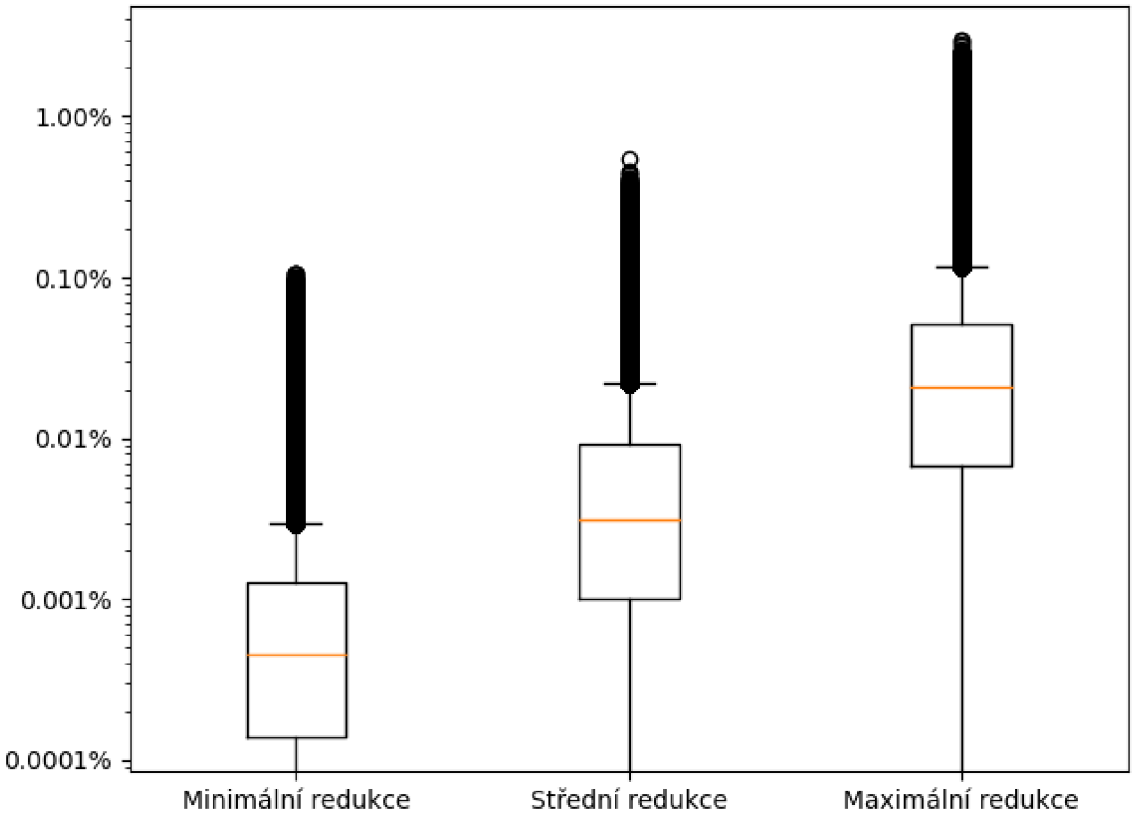


Obrázek 6.14: Odchylka vrstvy s ohniskem - Střední redukce - PH1-BM9-SC1



Obrázek 6.15: Odchylka vrstvy s ohniskem - Maximální redukce - PH1-BM9-SC1

Na obrázku 6.16 lze vidět krabicové grafy zobrazující normalizované chyby vůči ohnisku napříč výslednou maticí tlaku pro dataset `PH1-BM9-SC1`. Pro minimální redukci je graf prakticky totožný jako u předchozích datasetů. Pro střední redukce lze pozorovat drobný nárůst chyb ve srovnání s předchozími testy. Výrazné rozdíly lze sledovat pro testy s maximální redukcí přesnosti. Pro tu dosahují chyby přibližně třínásobné výše, a to nejen v případě maximální odchylky, ale i při porovnání hodnot prvního, druhého a třetího kvartilu.



Obrázek 6.16: Grafy normalizovaných odchylek vůči ohnisku - PH1-BM9-SC1

Kapitola 7

Závěr

Cílem této diplomové práce bylo optimalizovat výpočet simulace ultrazvukové neurostimulace prováděné softwarovým balíkem k-Wave ve verzi pro grafické karty implementované pomocí C++ s využitím platformy CUDA. Optimalizace mělo bylo docíleno využitím aritmetiky s redukovanou přesností. Toho se podařilo úspěšně dosáhnout s implementovanými třemi úrovněmi redukované přesnosti.

Implementované úrovně optimalizací se liší v redukovaných veličinách v závislosti na jejich vlivu na celkovou chybu výpočtů. Pro všechny úrovně však platí, že ohnisko tlaku zůstalo na zcela stejném místě. Pozice ohniska je přitom jedním z důležitých ukazatelů použitelnosti vytvořeného řešení.

Pro nejnižší úroveň optimalizací jsou veškeré sledované metriky chyb výpočtů zcela zanedbatelné. Přesto se podařilo ušetřit v některých testech i více než 10 % paměti GPU a přibližně 3,3 % z doby výpočtů testovaných simulací.

Při střední úroveň optimalizací jsou sledované metriky chyb o řád vyšší, nicméně v testovaných simulacích vznikají největší chyby mimo sledované médium. Lze tedy předpokládat, že i v tomto případě by chyby typicky neměly hrát významnou roli. Ušetřit se při této úrovni podařilo až 15,7 % paměti grafické karty a výpočty byly zkráceny zhruba o 6,5 % času.

Pro nejvyšší úroveň optimalizací jsou metriky chyb ještě o řád vyšší, přičemž chyby vznikají napříč celým sledovaným médiem, ve kterém je snaha indukovat tlak pomocí akustických vln. Hlavní otázkou tedy je, jak vysoká míra přesnosti je zapotřebí. Při porovnání s výsledky z jiných aplikací, které lze nalézt například v práci [9], se však i tato čísla jeví jako nikterak nerozumná. Paměti GPU se při této redukci podařilo ušetřit pro testované velké matice dat přes 21,5 %, výpočty se podařilo na stejných datasetech zkrátit o 8,4 %.

Všechny implementované úrovně optimalizace se tak z provedených testů jeví jako použitelné. Hlavním benefitem je poměrně razantní snížení paměťové náročnosti programu. Díky tomu je možné spouštět větší simulace, případně zvýšit přesnost pomocí jemnějšího vzorkování. Doba běhu byla zkrácena méně, než jaké bylo dosáhnuto redukce potřebné paměti. To je dáno tím, že redukce času bylo docíleno primárně na době běhu kernelů, které zabírají minoritní část výpočtů. Celkový čas výpočtů by mělo být teoreticky možné dále o něco zkrátit pomocí prokládaného ukládání akustické hustoty ve směru X ve směru Y do jedné matice hodnot. Tato změna by nicméně znamenala nutnost větších změn v implementaci. Do budoucna taky může být vhodné otestovat využití 8bitových datových typů pro ještě větší redukci. Pro to se nejvíce nabízí veličiny, které jsou redukovány již při minimální verzi optimalizací za využití datového typu `half`.

Literatura

[1] *Basic Hydrogen Properties* [online]. [cit. 2023-01-21]. Dostupné z: https://h2tools.org/hyarc/hydrogen-data/basic-hydrogen-properties.

[2] *Brain floating-point format (bfloat16)* [online]. [cit. 2022-12-28]. Dostupné z: https://en.wikichip.org/wiki/brain_floating-point_format.

[3] *Sound Speeds in Water, Liquid and Materials* [online]. [cit. 2023-01-20]. Dostupné z: https://www.rshydro.co.uk/sound-speeds/.

[4] ISO/IEC/IEEE International Standard - Floating-point arithmetic. *ISO/IEC 60559:2020(E) IEEE Std 754-2019*. 2020, s. 1–86. DOI: 10.1109/IEEESTD.2020.9091348. Dostupné z: https://ieeexplore.ieee.org/document/9091348.

[5] *FP8 Formats for Deep Learning* [online]. Září 2022 [cit. 2022-12-30]. Dostupné z: https://arxiv.org/abs/2209.05433v2.

[6] Al Onazi, M. M., Yurick, J. L., Harris, C., Nishimura, K., Suderman, K. et al. Therapeutic Ultrasound for Chemotherapy-Related Pain and Sensory Disturbance in the Hands and Feet in Patients With Colorectal Cancer: A Pilot Randomized Controlled Trial. *Journal of Pain and Symptom Management*. 2021, sv. 61, č. 6. ISSN 0885-3924. Dostupné z: https://doi.org/10.1016/j.jpainsymman.2020.10.028.

[7] Altun, a Sonkaya, A. The Most Common Side Effects Experienced by Patients Were Receiving First Cycle of Chemotherapy. *Iranian Journal of Public Health*. Srpen 2018, sv. 47, s. 1218–1219, [cit. 2023-05-08]. Dostupné z: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6123577/.

[8] Arblaster, J. W. *Densities of Osmium and Iridium* [online]. 1989 [cit. 2023-01-21]. Dostupné z: https://technology.matthey.com/article/33/1/14-16/.

[9] Aubry, J.-F., Bates, O., Boehm, C., Butts Pauly, K., Christensen, D. et al. Benchmark problems for transcranial ultrasound simulation: Intercomparison of compressional wave modelsa). *The Journal of the Acoustical Society of America*. Srpen 2022, sv. 152, č. 2, s. 1003–1019, [cit. 2023-05-08]. ISSN 0001-4966. Dostupné z: https://doi.org/10.1121/10.0013426.

[10] Blackmore, J., Shrivastava, S., Sallet, J., Butler, C. R. a Cleveland, R. O. Ultrasound Neuromodulation: A Review of Results, Mechanisms and Safety. *Ultrasound in Medicine & Biology*. Elsevier. 2019, sv. 45, č. 7, [cit. 2023-05-08]. DOI: 10.1016/j.ultrasmedbio.2018.12.015. Dostupné z: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6996285/.

[11] Corporation, N. *CUDA Toolkit* [online]. [cit. 2023-05-15]. Dostupné z: https://developer.nvidia.com/cuda-toolkit.

[12] Corporation, N. *NVIDIA Tesla P100* [online]. 2016 [cit. 2023-01-01]. Dostupné z: https://images.nvidia.com/content/pdf/tesla/whitepaper/pascal-architecture-whitepaper-v1.2.pdf.

[13] Corporation, N. *NVIDIA TURING GPU ARCHITECTURE* [online]. Září 2018 [cit. 2023-01-01]. Dostupné z: https://images.nvidia.com/aem-dam/en-zz/Solutions/design-visualization/technologies/turing-architecture/NVIDIA-Turing-Architecture-Whitepaper.pdf.

[14] Corporation, N. *NVIDIA AMPERE GA102 GPU ARCHITECTURE Second-Generation RTX* [online]. 2020 [cit. 2022-12-28]. Dostupné z: https://www.nvidia.com/content/PDF/nvidia-ampere-ga-102-gpu-architecture-whitepaper-v2.1.pdf.

[15] Corporation, N. *NVIDIA V100 TENSOR CORE GPU* [online]. Leden 2020 [cit. 2023-05-17]. Dostupné z: https://images.nvidia.com/content/technologies/volta/pdf/volta-v100-datasheet-update-us-1165301-r5.pdf.

[16] Corporation, N. *CUDA C++ Best Practices Guide* [online]. Prosinec 2022 [cit. 2023-01-02]. Dostupné z: https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html.

[17] Corporation, N. *NVIDIA H100 Tensor Core GPU Architecture* [online]. 2022 [cit. 2023-01-01]. Dostupné z: https://resources.nvidia.com/en-us-tensor-core/gtc22-whitepaper-hopper.

[18] Díaz Alejo, J. F., González Gómez, I. a Earl, J. Ultrasounds in cancer therapy: A summary of their use and unexplored potential. *Oncology Reviews*. PAGEPress Publications. Únor 2022, sv. 16, č. 1, [cit. 2023-05-08]. DOI: 10.4081/oncol.2022.531. Dostupné z: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8941342/.

[19] Higham, N. *Half Precision Arithmetic: fp16 Versus bfloat16* [online]. Prosinec 2018 [cit. 2022-12-26]. Dostupné z: https://nhigham.com/2018/12/03/half-precision-arithmetic-fp16-versus-bfloat16/.

[20] Ho, N. a Wong, W. *Exploiting half precision arithmetic in Nvidia GPUs* [online]. [cit. 2023-01-01]. Dostupné z: https://www.comp.nus.edu.sg/~wongwf/papers/hpec17.pdf.

[21] IBM Cloud Education, I. C. E. *Learn how green computing reduces energy consumption and lowers carbon emissions from the design, use and disposal of technology products.* [online]. Duben 2022 [cit. 2023-05-15]. Dostupné z: https://www.ibm.com/cloud/blog/green-computing.

[22] K. Naugolnykh, L. O. *Nonlinear Wave Processes in Acoustics*. Cambridge University Press, květen 1998. 4 s. ISBN 9780521399845. Dostupné z: https://books.google.cz/books?id=0aBgQgAACAAJ.

[23] Kamimura, H. A. S., Conti, A., Toschi, N. a Konofagou, E. E. Ultrasound Neuromodulation: Mechanisms and the Potential of Multimodal Stimulation for

Neuronal Function Assessment. *Frontiers in Physics*. 2020, sv. 8. DOI: 10.3389/fphy.2020.00150. ISSN 2296-424X. Dostupné z: https://www.frontiersin.org/articles/10.3389/fphy.2020.00150.

[24] Luitjens, J. *CUDA Pro Tip: Increase Performance with Vectorized Memory Access* [online]. Listopad 2013 [cit. 2023-05-15]. Dostupné z: https://developer.nvidia.com/blog/cuda-pro-tip-increase-performance-with-vectorized-memory-access/.

[25] Narasimhan, S. *NVIDIA, Arm, and Intel Publish FP8 Specification for Standardization as an Interchange Format for AI* [online]. Srpen 2022 [cit. 2023-05-15]. Dostupné z: https://developer.nvidia.com/blog/nvidia-arm-and-intel-publish-fp8-specification-for-standardization-as-an-interchange-format-for-ai/.

[26] Patterson, D. *The Top 10 Innovations in the New NVIDIA Fermi Architecture, and the Top 3 Next Challenges* [online]. Září 2009 [cit. 2023-01-01]. Dostupné z: https://www.nvidia.com/content/PDF/fermi_white_papers/D.Patterson_Top10InnovationsInNVIDIAFermi.pdf.

[27] Pearce, A., Haas, M., Viney, R., Pearson, S. A., Haywood, P. et al. Incidence and severity of self-reported chemotherapy side effects in routine care: A prospective cohort study. *PLoS One*. Oct 2017, sv. 12, č. 10, s. e0184360, [cit. 2023-05-08]. DOI: 10.1371/journal.pone.0184360. Dostupné z: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5634543/.

[28] Smith, R. *NVIDIA Announces H100 NVL - Max Memory Server Card for Large Language Models* [online]. [cit. 2023-05-17]. Dostupné z: https://www.anandtech.com/show/18780/nvidia-announces-h100-nvl-max-memory-server-card-for-large-language-models.

[29] Treeby, B. E. a Cox, B. T. k-Wave: MATLAB toolbox for the simulation and reconstruction of photoacoustic wave fields. *Journal of Biomedical Optics*. SPIE. 2010, sv. 15, č. 2, [cit. 2023-05-08]. Dostupné z: https://doi.org/10.1117/1.3360308.

[30] Treeby, B. E., Jaros, J., Rendell, A. P. a Cox, B. T. Modeling nonlinear ultrasound propagation in heterogeneous media with power law absorption using a k-space pseudospectral method. *The Journal of the Acoustical Society of America*. Červen 2012, sv. 131, č. 6, [cit. 2023-05-08]. ISSN 0001-4966. Dostupné z: https://doi.org/10.1121/1.4712021.