VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
DEPARTMENT OF COMPUTER SYSTEMS

AKCELERACE ULTRAZVUKOVÝCH SIMULACÍ POMOCÍ MULTI-GPU SYSTÉMŮ

ACCELERATION OF ULTRASONIC SIMULATIONS USING MULTI-GPU SYSTEMS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE **Bc. MARTIN STODŮLKA**
AUTHOR

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

BRNO 2021

Brno University of Technology
Faculty of Information Technology

Department of Computer Systems (DCSY) Academic year 2020/2021

Master's Thesis Specification

Student: **Stodůlka Martin, Bc.**
Programme: Information Technology and Artificial Intelligence
Specialization: High Performance Computing
Title: **Acceleration of Ultrasound Simulations on Multi-GPU Systems**
Category: Parallel and Distributed Computing
Assignment:

1. Seznamte se s ultrazvukovým simulátorem k-Wave a jeho akcelerovanou implementací určenou pro systémy vybavené jednou grafickou kartou.
2. Osvojte si pokročilé techniky akcelerace vědeckých aplikací na systémech s více grafickými kartami. Zaměřte se především na technologii CUDA a její unifikovanou paměť.
3. Implementujte prototypovou aplikaci využívající několika grafických karet za účelem měření výkonnosti, propustnosti a latence přístupu do lokálních i vzdálených paměti.
4. Navrhněte algoritmus pro výpočet ultrazvukových simulací na více GPU, zaměřte se především na efektivní výpočet Fourierových transformací a sběru simulačních dat.
5. Navržené řešení implementujte a otestujte na reálné sadě simulačních úloh.
6. Zhodnoťte dosažené výsledky a diskutujte přínos navržené implementace pro řešení realistických ultrazvukových simulací.

Recommended literature:

- Dle pokynů vedoucího.

Requirements for the semestral defence:

- Body 1 až 3 zadání.

Detailed formal requirements can be found at https://www.fit.vut.cz/study/theses/

Supervisor: **Jaroš Jiří, doc. Ing., Ph.D.**
Head of Department: Sekanina Lukáš, prof. Ing., Ph.D.
Beginning of work: November 1, 2020
Submission deadline: May 19, 2021
Approval date: April 22, 2021

Abstrakt

V této práci je kladen důraz na multi-GPU systémy a využití CUDA unifikované paměti. Hlavním cílem je akcelerovat výpočet 3D FFT, který je hlavní součástí simulací knihovny k-Wave. K-Wave je C++/Matlab knihovna určena pro simulaci šíření ultrazvukových vln v 1D, 2D nebo 3D prostoru. Akcelerace těchto funkcí je potřebná, jelikož se jedná o výpočetně náročně simulace.

Abstract

The main focus of this project is usage of multi-GPU systems and usage of CUDA unified memory. Its goal is to accelerate computation of 2D and 3D FFT, which is the main part of simulations in k-Wave library. K-Wave is a C++/Matlab library used for simulations of propagation of ultrasonic waves in 1D, 2D or 3D space. Acceleration of these functions is necessary, because the simulations are computationally intensive.

Klíčová slova

C++, CUDA, unifikovaná paměť, k-Wave, GPU, multi-GPU, NVLink, FFT, cuFFT, simulace, ultrazvuk, SC-GPU1, It4i, Barbora

Keywords

C++, CUDA, unified memory, k-Wave, GPU, multi-GPU, NVLink, FFT, cuFFT, simulation, ultrasonic, SC-GPU1, It4i, Barbora

Citace

STODŮLKA, Martin. *Akcelerace ultrazvukových simulací pomocí multi-GPU systémů*. Brno, 2021. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce doc. Ing. Jiří Jaroš, Ph.D.

Akcelerace ultrazvukových simulací pomocí multi-GPU systémů

Prohlášení

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

.....
Martin Stodůlka
25. května 2021

Poděkování

Chtěl bych poděkovat panu doc. Ing. Jiřímu Jarošovi za spolupráci a konzultace i během náročného období pandemie.

Obsah

1 Úvod **3**

2 Multi-GPU a unifikovaná paměť **4**

3 Testování unifikované paměti **6**
3.1 Rychlost unifikované paměti 6
3.2 2D Filtr 6
3.3 Broadcast 7
3.4 Transponování 2D matic 7
3.5 CufftXt 2D a 3D 7

4 Výsledky předběžného testování **9**
4.1 Konfigurace použitých zařízení 9
4.1.1 Barbora 9
4.1.2 SC-GPU1 10
4.2 Výsledky 10
4.2.1 Rychlost unifikované paměti 10
4.2.2 2D Filtr 11
4.2.3 Broadcast 12
4.2.4 Transponování 13
4.2.5 CufftXt2D 14
4.2.6 CufftXt3D 15
4.3 Závěr předběžného testování 16

5 Implementace v knihovně k-Wave **17**
5.1 k-Wave CUDA 17
5.1.1 kSpaceFirstOrderSolver 17
5.1.2 Matrix třídy 17
5.1.3 Pomocné kernely 18
5.2 Model výpočtu 18
5.3 Měření referenčních hodnot 19

6 Převod na unifikovanou paměť **21**
6.1 Testování unifikované paměti 21
6.2 Speciální vlastnost unifikované paměti 24

7 Multi-GPU implementace **25**
7.1 CuFFT implementace 25

7.1.1 Implementace . . . 25
7.1.2 Měření . . . 26
7.2 CuFFT implementace používající prefetch . . . 30
7.2.1 Implementace . . . 30
7.2.2 Měření . . . 30
7.3 CuFFTXt . . . 33
7.3.1 Implementace . . . 33
7.3.2 Měření . . . 34

8 Závěr **37**

Literatura **38**

A Obsah přiloženého paměťového média **39**

B Manuál **40**
B.1 Přeložení programu . . . 40
B.2 Nastavení počtu GPU . . . 40
B.3 Spuštění programu . . . 40

Kapitola 1

Úvod

Tato práce se zabývá využitím multi-GPU systémů pro akceleraci výpočtu 3D FFT, které jsou hlavní součástí výpočtů ultrazvukových simulací knihovny k-Wave[6]. Funkce této knihovny již jsou akcelerované, ale pouze na jediném GPU. Akcelerace pomocí více GPU je dalším logickým krokem vývoje této knihovny. Zvýšení výkonu je zde nutné, jelikož se jedná o výpočetně náročné simulace. Cílem této práce je zjistit výkon a případné využití unifikované paměti[3] a cuFFT[1] knihovny.

V druhé kapitole jsou krátce uvedeny teoretické základy ohledně multi-GPU a konceptu unifikované paměti[3][5] CUDA.

V následujících dvou kapitolách budou uvedeny popisy prostředků použitých pro akceleraci. Dále budou uvedeny postupy použité pro předběžné zjištění rychlosti a použitelnosti prostředků použitých pro akceleraci, včetně použitých zařízení na kterých byla měření prováděna. V druhé kapitole jsou uvedeny a shrnuty výsledky předběžného měření.

V páté kapitole je stručně popsána původní implementace knihovny k-Wave[6]. K implementaci jsou uvedeny měření rychlosti výpočtu pro srovnání s implementacemi pro urychlení výpočtu.

V posledních dvou kapitolách jsou uvedeny různé implementace pro urychlení výpočtu a je zhodnoceno jejich zrychlení pomocí grafů silného a slabého škálování. Z výsledků měření je každá metoda na závěr zhodnocena.

Na závěr jsou výsledky práce zhodnoceny a je navrhnut další možný postup v pokračování této práce.

Kapitola 2

Multi-GPU a unifikovaná paměť

Před použitím více GPU je dobré porozumět jak výpočty na více GPU fungují, kdy se vyplatí počítat na více GPU oproti jednomu GPU a případně čím je počítání omezeno.

Pro triviálně paralelizovatelnou úlohu, kde při dekompozici na podúlohy není potřeba žádné komunikace mezi GPU, se algoritmy nějak zvlášť neliší od algoritmu na jednom GPU. Data z CPU jsou rozdělena mezi jednotlivá GPU a následně jejich výsledky shromážděny zpět na CPU.

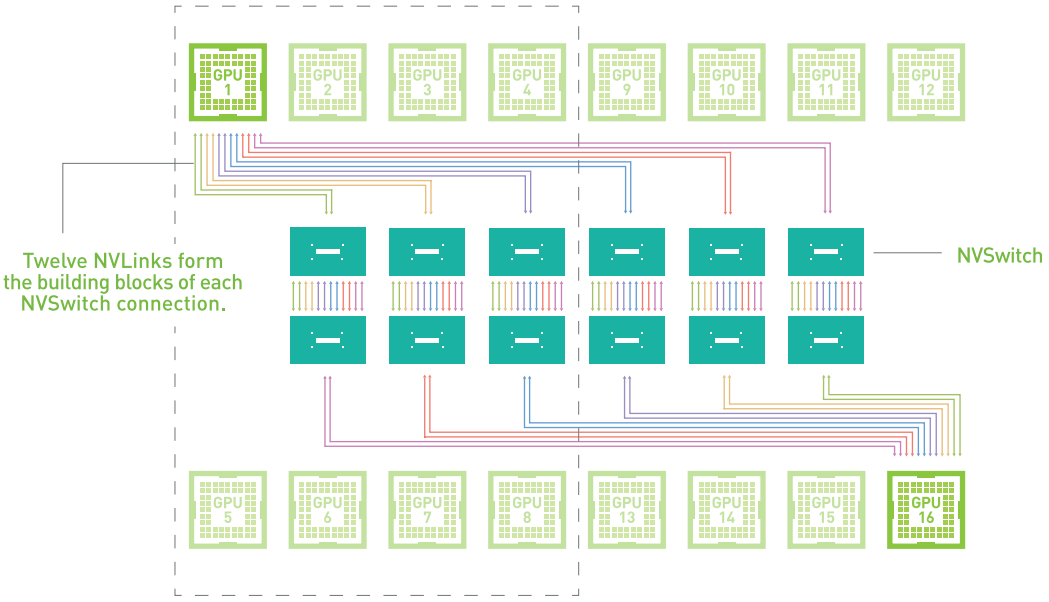
Problémy nastávají, když během výpočtu jednotlivé GPU potřebují mezi sebou nebo s CPU komunikovat. Běžně by na tuto komunikaci bylo využito rozhraní PCI-e, ovšem takovéto rozhraní je limitováno latencí a rychlostí. Rozhraní PCI-e je mnohonásobně pomalejší než propojení GPU jádra a globální paměti GPU. Pro srovnání propustnost jedné linky PCI-e 3.0 je 1 GB/s, kdežto propustnost GDDR5 na jednom paměťovém čipu je 32 GB/s. Z tohoto důvodu byl zaveden NVLink[2].

PCIe 1.x	∼0.25 GB/s
PCIe 2.x	∼0.5 GB/s
PCIe 3.x	∼1 GB/s
PCIe 4.0	∼2 GB/s
PCIe 5.0	∼4 GB/s
NVLink 1.0	∼2.5 GB/s
NVLink 2.0	∼3.125 GB/s
NVLink 3.0	∼6.25 GB/s

Tabulka 2.1: Rychlosti NVLink a PCI-e pro jednu linku v jednom směru[2]

V podstatě se jedná o přímé propojení více GPU jednotek mezi sebou. Výhodou oproti PCI-e je decentralizace komunikace, kde každé GPU může přímo komunikovat s jiným a s vyšší rychlostí.

NVLink[2] běžně umožňuje propojit nejvýše čtyři GPU. NSwitch je rozšíření NVLinku[2]. Pomocí switchů propojuje 8 až 16 GPU. NVSwitch je zatím pouze použit v počítačích DGX-2 pro umělou inteligenci.



Obrázek 2.1: **NVSwitch** Diagram propojení maximálního počtu (16) GPU pomocí technologie NVSwitch. Převzato z: https://www.nvidia.com/en-us/data-center/nvlink/

I s těmito všemi zapojeními pořád zůstává problém řešení komunikace mezi GPU. V normální situaci by se komunikace musela řešit kopírováním dat z jednoho GPU do druhého voláním rutiny ze strany CPU. Efektivita komunikace zde čistě závisí na programátorovi a jeho implementaci.

Pro zjednodušení implementace a zrychlení výměny dat mezi jednotlivými GPU a CPU byl navržen sdílený paměťový prostor neboli unifikovaná paměť[3][5] v jazyce CUDA. Unifikovaná paměť[3][5] je alokovaná pomocí speciálních funkcí z CUDA prostředí. Z pohledu C++ se jedná o normální ukazatel na pole, ale při přístupu do tohoto pole z CPU nebo GPU je hardwarově nebo softwarově zaručeno migrování (paměťových) stránek na příslušné zařízení. Od architektury Pascal a výše je tento mechanismus implementován hardwarově. O správu a migraci stránek se stará driver grafické karty.

Kapitola 3

Testování unifikované paměti

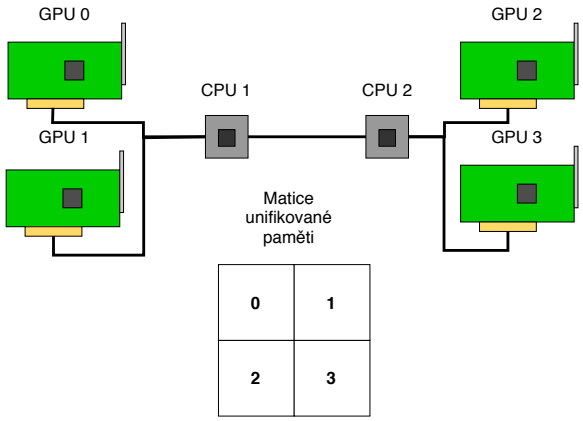
V následující kapitole budou popsány jednotlivé testy pro změření předběžného výkonu unifikované paměti[3][5] na více GPU a cuFFTXt[1]. Všechny testy byly implementovány jako samostatný program mimo k-Wave[6] v jazyce C++ s použitím CUDY. U všech testů je čas měřen pomocí standardní C++ knihovny `chrono`.

3.1 Rychlost unifikované paměti

Rychlost unifikované paměti[3][5] byla testována nad jedním velkým paměťovým blokem. Paměťový blok je neinicializovaný, jde o nově vytvořený blok v paměti. Paměťové přenosy byly testovány: z unifikované paměti[3][5] do unifikované paměti[3][5], z CPU paměti do unifikované paměti[3][5], z GPU paměti do unifikované paměti[3][5] a z unifikované paměti[3][5] do GPU paměti. Ve výsledcích nebylo nezahrnováno první měření, jelikož při práci s unifikovanou pamětí[3][5] dochází k výpadkům stránek při prvním průchodu.

3.2 2D Filtr

U 2D filtru byla testována rychlost unifikované paměti[3][5] u jedné z nejčastějších úloh na GPU. Konkrétně bylo počítáno filtrovaní matice typu `float` maticí o rozměrech 3x3 stejného typu. Jádro mělo náhodně inicializované hodnoty. Úloha byla rozmístěna mezi více GPU po dlaždicích stejných rozměrů, s tím že pro každé GPU se sériově spouštěl kernel s jinými parametry z CPU. Pro vyzkoušení vlivu sousednosti GPU na rychlost výpočtu, byl test spuštěn několikrát s odlišným rozděleným dlaždic mezi GPU.



Obrázek 3.1: Názorná ukázka důvodu, proč bylo testováno rozdělení dlaždic mezi GPU několika způsoby. Například kdyby 0. a 3. dlaždice byla počítána na GPU 0 a 1 a 1. a 2. dlaždice na GPU 2 a 3, tak by docházelo výhradně ke komunikacím přes dva CPU sokety, které by test značně zpomalily.

3.3 Broadcast

U testu broadcast bylo testováno kopírování bloku dat z unifikované paměti[3][5] na všechna GPU. Broadcast byl implementován pomocí `MemCpyAsync`, kde pro každé kopírování byl spuštěn ve svém CUDA streamu a na konci se všechny CUDA streamy synchronizovali. Čas byl změřen po synchronizaci všech CUDA streamů.

3.4 Transponování 2D matic

U transponování 2D matic byla vyzkoušena rychlost komunikace mezi jednotlivými GPU v úloze náročné na paměťovou komunikaci. Úloha byla rozdělena mezi GPU po horizontálních proužcích paměti. Každý kernel byl spuštěn v individuálním CUDA streamu. Po spuštění všech kernelů byly CUDA streamy synchronizovány a následně změřen výsledný čas. Samotná transpozice byla implementována s použitím sdílené paměti pro optimalizace paměťových transakcí. Stejně jako u 2D filtru bylo rozdělení pamětí mezi GPU testováno několika způsoby.

3.5 CufftXt 2D a 3D

Tento test sloužil pouze k otestování knihovních funkcí CUDA pro FFT na více GPU. Bohužel tyto funkce nepodporují použití unifikované paměti[3][5], jelikož používají vlastní funkce na alokaci a inicializaci paměti.

Pro použití cufftXt[1] funkcí je nejdříve vytvořit a inicializovat plán pomocí `cufftCreate` a `cufftMakePlan` (1D, 2D atd.) Dále je potřeba alokovat a inicializovat paměť pomocí `cufftXtMalloc` a `cufftXtMemcpy`, které vracejí deskriptory které udávají rozložení paměti na GPU a samotné paměťové bloky. Knihovna cufftXt[1] si rozdělení paměti řídí sama. Po vytvoření plánu a inicializaci paměti stačí spustit daný plán nad deskriptorem paměti pomocí `cufftXtExecDescriptor` následovaný nějakou možnou variantou (C2C, Z2Z, R2C

atd.) Čas byl měřený pouze pro `cufftXtExecDescriptor`, inicializace plánu a paměti do měření nebyla zahrnuta.

Pro tyto testy byla otestována varianta `C2C, CUFFT_FORWARD` na matici inicializované jedničkami $1 + 0 \cdot i$.

Kapitola 4

Výsledky předběžného testování

V následující kapitole budou ukázány naměřené hodnoty testů zmíněných v předchozí kapitole. Každý test byl opakován 11krát. Výsledná hodnota byla získána jako průměr posledních deseti hodnot. Testy byly měřeny na dvou zařízeních: počítač SC-GPU1 na fakultě FIT a jeden z uzlů superpočítače Barbory v Ostravě vybavený GPU. Testování bylo provedeno pro výpočty na 1 GPU a 4 GPU.

4.1 Konfigurace použitých zařízení

Níže jsou uvedeny tabulky popisující konfigurace SC-GPU 1 a superočítače Barbora, které byly použity pro testování.

4.1.1 Barbora

Operační systém	Linux
RAM (GB)	192
CPU	2x Intel Skylake Gold 6126, 2.6 GHz
GPU	4x NVIDIA Tesla V100-SXM2
GPU propojení	NVLINK

Tabulka 4.1: Konfigurace akcelerovaného uzlu Barbory

Architektura	Volta
počet CUDA jader	5120
GPU Clock	1246 MHZ
GPU Boost Clock	1380 MHZ
Single Precision výkon	15.7 TFLOPS
Double Precision výkon	7.8 TFLOPS
Typ paměti	HBM2
Velikost paměti	16 GB
Velikost paměťové sběrnice	384 bit
Propustnost paměti	900 GB/S
Příkon	300 W

Tabulka 4.2: Parametry NVIDIA Tesla V100-SXM2

4.1.2 SC-GPU1

Operační systém	Linux
RAM (GB)	64
CPU	2x Intel Xeon CPU E5-2620 v3 @ 2.40GHz
GPU	4x NVIDIA GTX 1080
GPU propojení	PCI-e

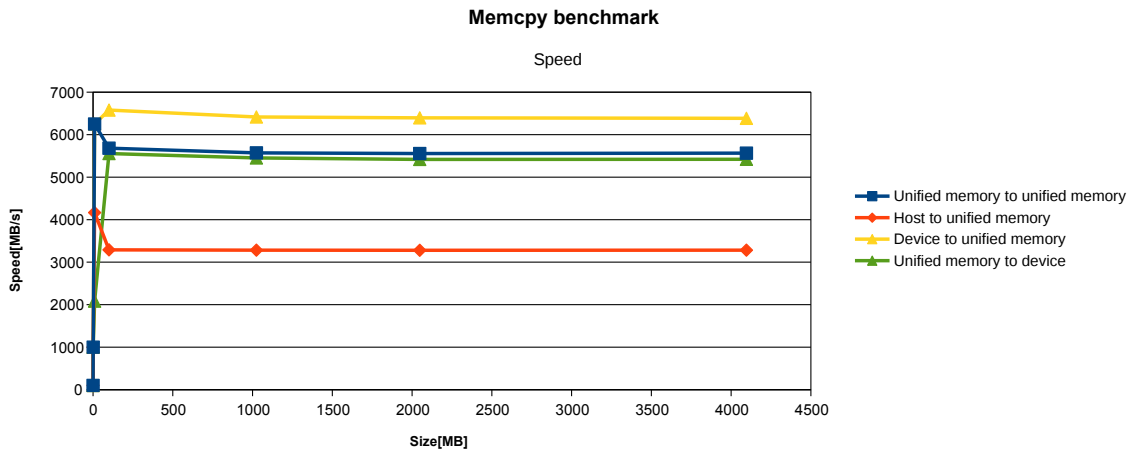
Tabulka 4.3: Konfigurace SC-GPU1

Architektura	Pascal
počet CUDA jader	2560
GPU Clock	1607 MHZ
GPU Boost Clock	1733 MHZ
Single Precision výkon	8.9 TFLOPS
Double Precision výkon	0.277 TFLOPS
Typ paměti	GDDR5X
Velikost paměti	8 GB
Velikost paměťové sběrnice	256 bit
Propustnost paměti	320 GB/S
Příkon	180 W

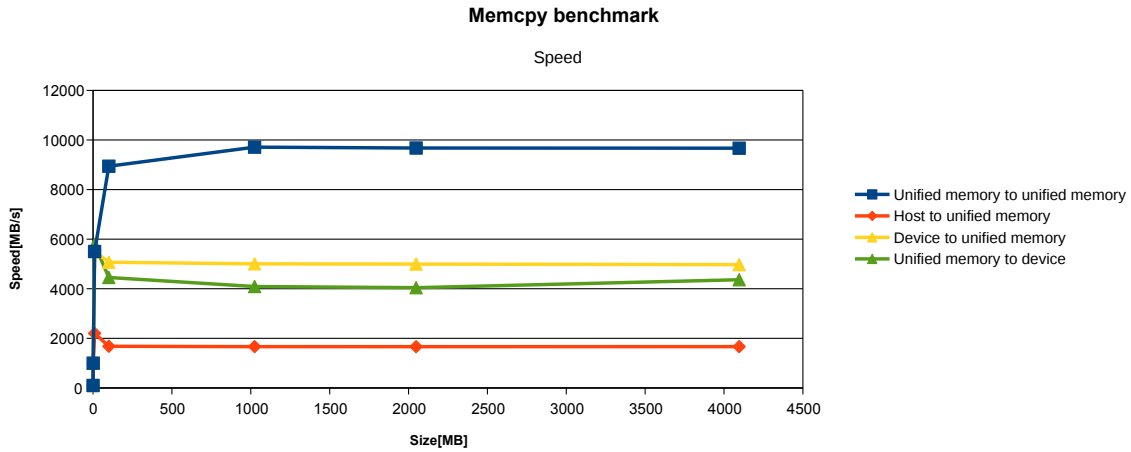
Tabulka 4.4: Parametry NVIDIA GTX 1080

4.2 Výsledky

4.2.1 Rychlost unifikované paměti



Obrázek 4.1: Výsledky rychlosti unifikované paměti na Barboře

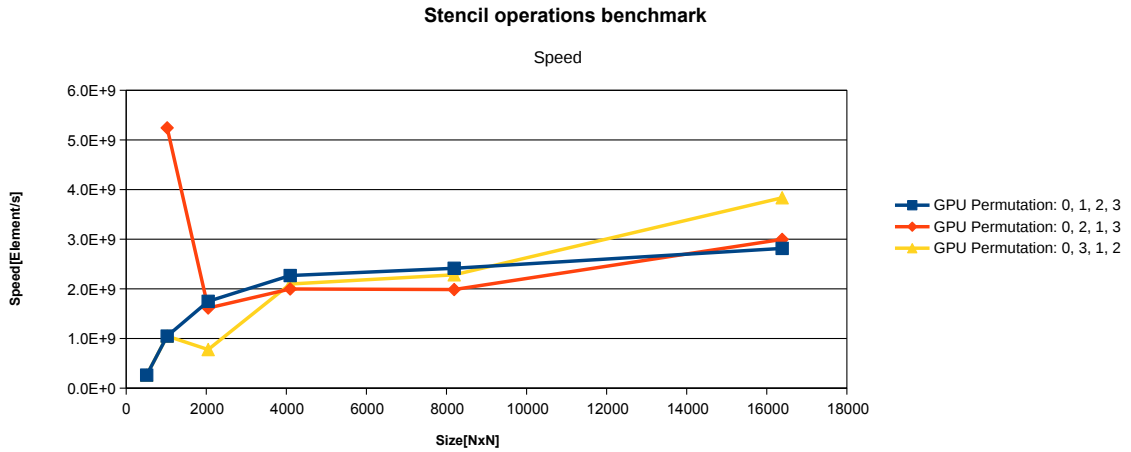


Obrázek 4.2: Výsledky rychlosti unifikované paměti na SC-GPU1

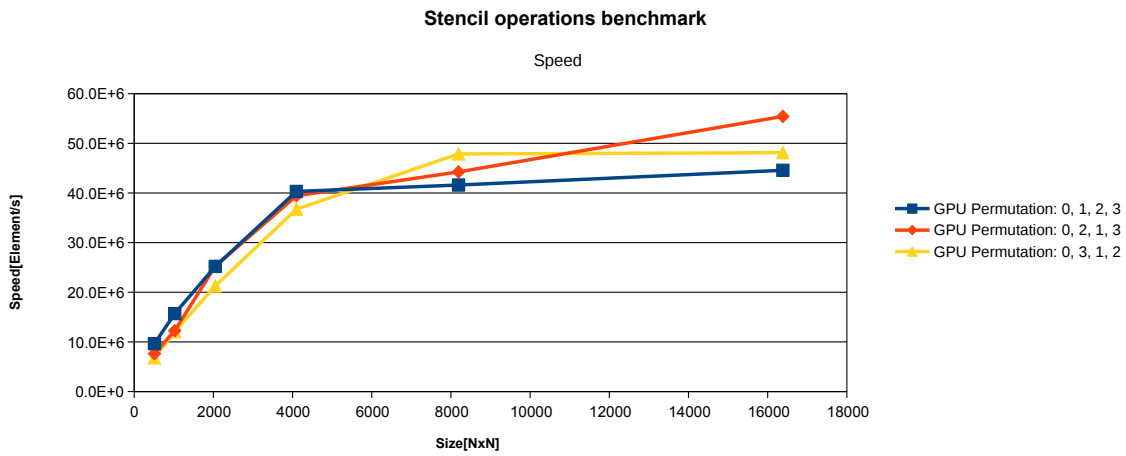
Rychlost paměti byla testována na velikostech: 0.1, 1, 10, 100, 1024, 2048 a 4096 MB. Výsledky u 0.1 a 1 MB byly menší jak 1 milisekunda a nešlo je přesněji změřit, tudíž jsou pevně nastavené na 1 milisekundu.

Jak na SC-GPU1 tak na Barboře bylo kopírování z CPU do unifikované paměti[3][5] nejpomalejší, což by se dalo očekávat. Zvláštní je, že kopírovaní z unifikované paměti[3][5] do unifikované paměti[3][5] na SC-GPU1 bylo podstatně rychlejší než ostatní metody a dokonce i Barbora, i přestože má (co se týče paměti) výkonnější GPU.

4.2.2 2D Filtr



Obrázek 4.3: Výsledky 2D filtru na Barboře



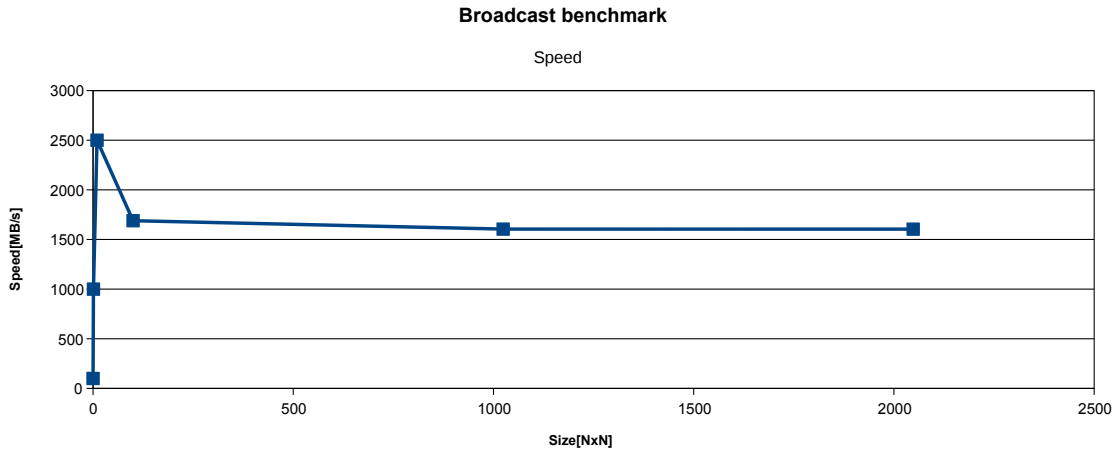
Obrázek 4.4: Výsledky 2D filtru na SC-GPU1

Filtrování bylo testováno na rozměrech matice: 512x512, 1024x1024, 2048x2048, 4096x4096, 8192x8192 a 16384x16384. Výsledky u 512x512 a 1024x1024 byly menší jak 1 milisekunda a nešlo je přesněji změřit, tudíž jsou pevně nastavené na 1 milisekundu.

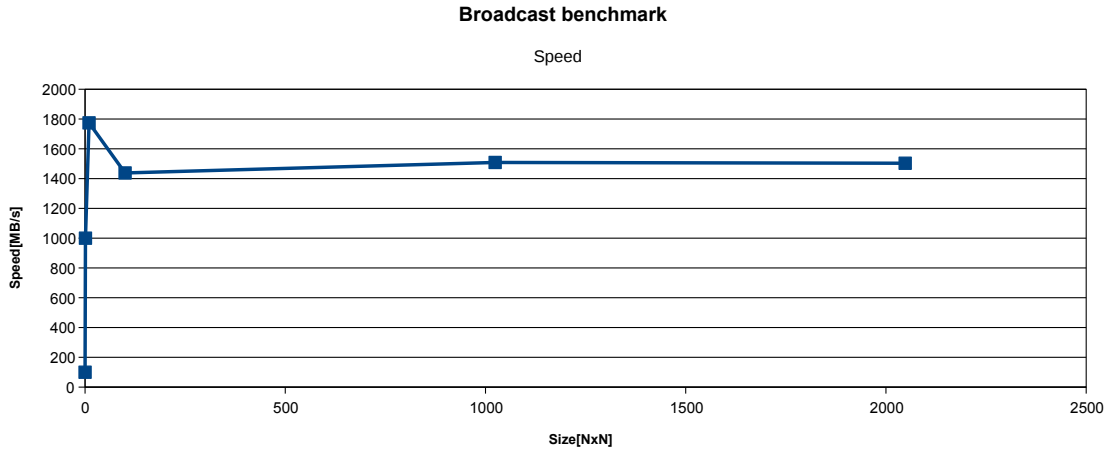
Barbora byla výrazně rychlejší než SC-GPU1 (až 60 krát), což se dalo očekávat, jelikož má mnohem výkonnější GPU. Rozprostření dlaždic mezi různé permutace GPU nehrálo moc významnou roli. U Barbory permutace 0,3,1,2 sice dává lepší výsledky, ale otázkou je, jestli to není malým počtem měření, jelikož tato permutace u menších velikostí není lepší než ostatní. Tohle může být také zapříčiněno tím, že u 2D filtrování 3x3 jádrem dochází ke komunikaci pouze na malém úseku matice (sousedící okraje dlaždic).

Ve srovnání s implementací pro 1 GPU byla implementace pro 4 GPU výrazně pomalejší (až 100x). Tohle může být způsobeno tím, že na okrajích každé dlaždice jsou způsobeny výpadky stránek, které vedou k velké ztrátě výkonu.

4.2.3 Broadcast



Obrázek 4.5: Výsledky Barbory

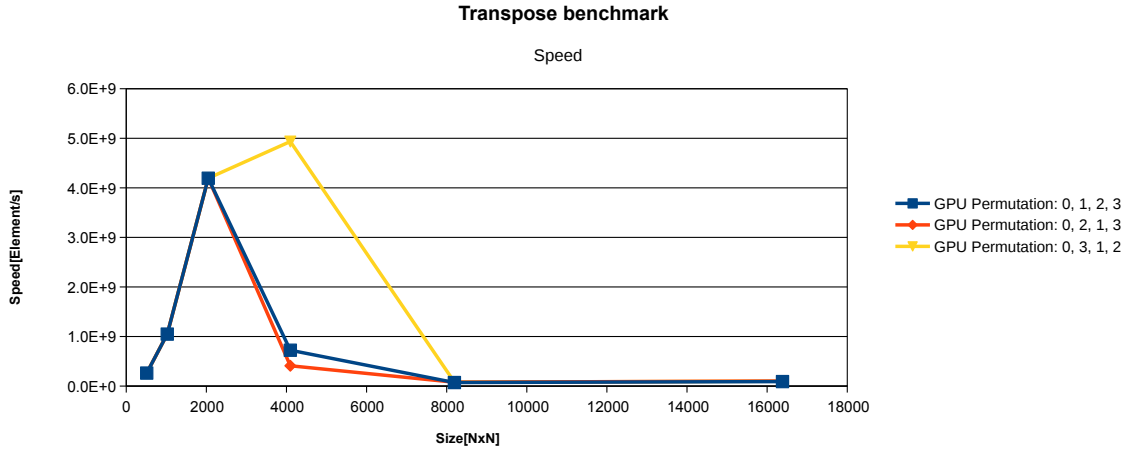


Obrázek 4.6: Výsledky SC-GPU1

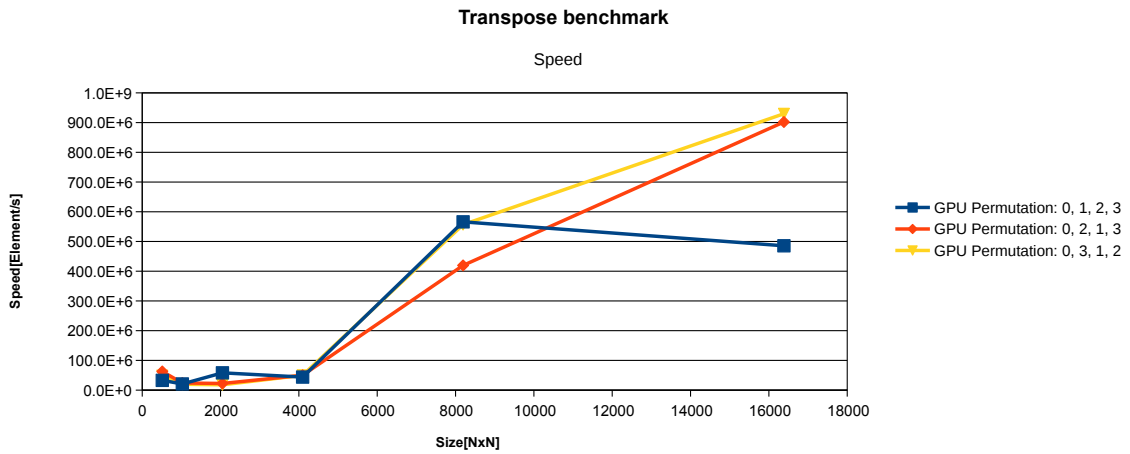
Broadcast byl testován na velikostech: 0.1, 1, 10, 100, 1024 a 2048 MB. Výsledky u 0.1 a 1 MB byly menší jak 1 milisekunda a nešlo je přesněji změřit, tudíž jsou pevně nastavené na 1 milisekundu.

Zde je důležité si uvědomit, že rychlost je vypočítána jako $\frac{velikost}{celkovy\_cas}$, tudíž se vztahuje k jednomu GPU. Ve skutečnosti jsou data kopírována na 4 GPU současně. Překvapivě Barbora nebyla o moc rychlejší než SC-GPU1, i přesto že má podstatně rychlejší paměť. Je možné, že při kopírování se paměťové stránky nacházejí v paměti CPU která se stává úzkým hrdlem.

4.2.4 Transponování



Obrázek 4.7: Výsledky transponování matic na Barboře



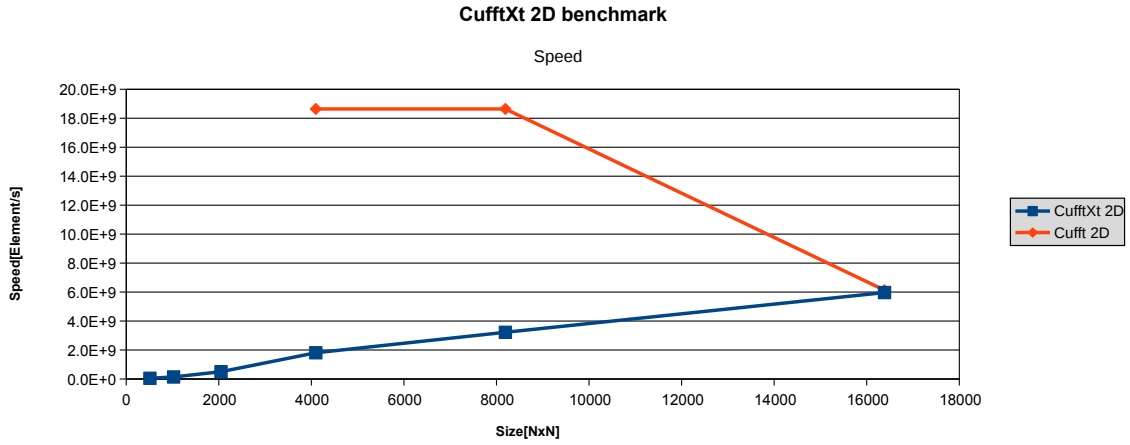
Obrázek 4.8: Výsledky transponování matic na SC-GPU1

Transponovaní matic bylo testováno na velikostech: 512x512, 1024x1024, 2048x2048, 4096x4096, 8192x8192 a 16384x16384. U Barbory pro velikost 512x512, 1024x1024 a 2048x2048 byly časy menší jak 1 milisekunda a nešlo je přesněji změřit, tudíž jsou pevně nastavené na 1 milisekundu.

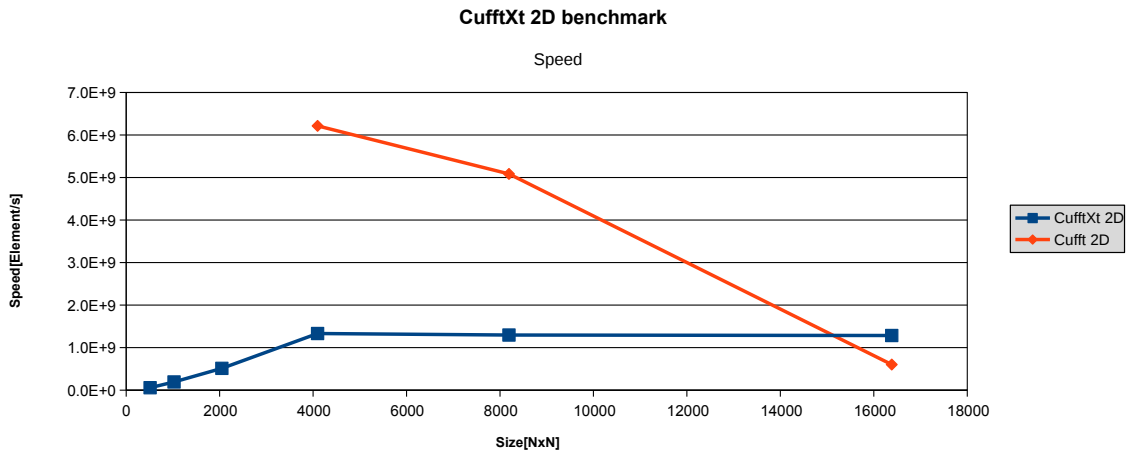
Zde docházelo k velice zvláštním výsledkům. Obzvlášť z výsledků Barbory lze vidět velmi pomalou rychlost u velikostí větších jak 4096x4096. Implementace transponování byla zkontrolována a ověřena, že vrací správné výsledky. Po zkoumání profilovacích výpisů `nvprof` bylo zjištěno, že docházelo k podstatněji větším výpadkům stránek a novým komunikacím, které u nižších velikostí nebyly.

Ve srovnání s implementací pro 1 GPU byla implementace pro 4 GPU výrazně pomalejší (až 1000x). Tohle může být způsobeno tím že na okrajích každé dlaždice nebo při přístupu do výstupní matice jsou způsobeny výpadky stránek, které vedou k velké ztrátě výkonu.

4.2.5 CufftXt2D



Obrázek 4.9: Výsledky CufftXt2D na Barboře



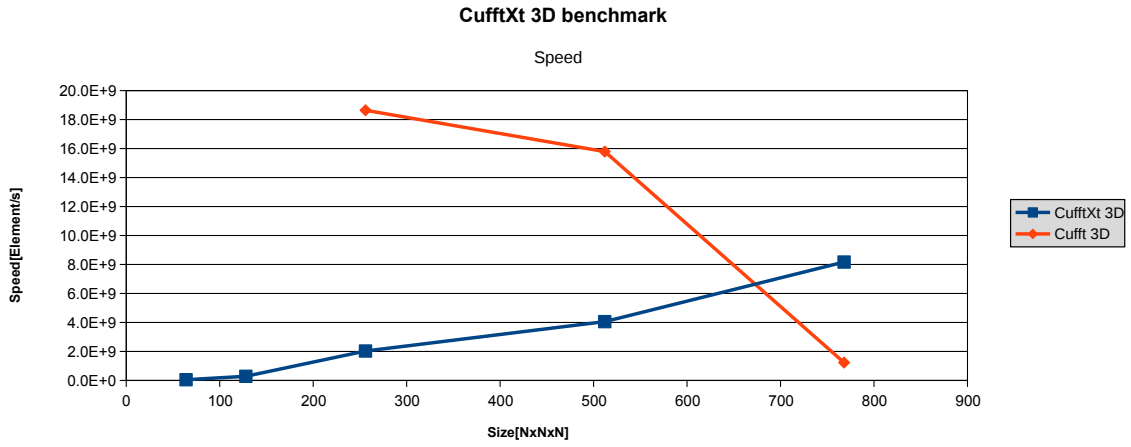
Obrázek 4.10: Výsledky CufftXt2D na SC-GPU1

CufftXt2D bylo testováno na velikostech: 512x512, 1024x1024, 2048x2048, 4096x4096, 8192x8192 a 16384x16384.

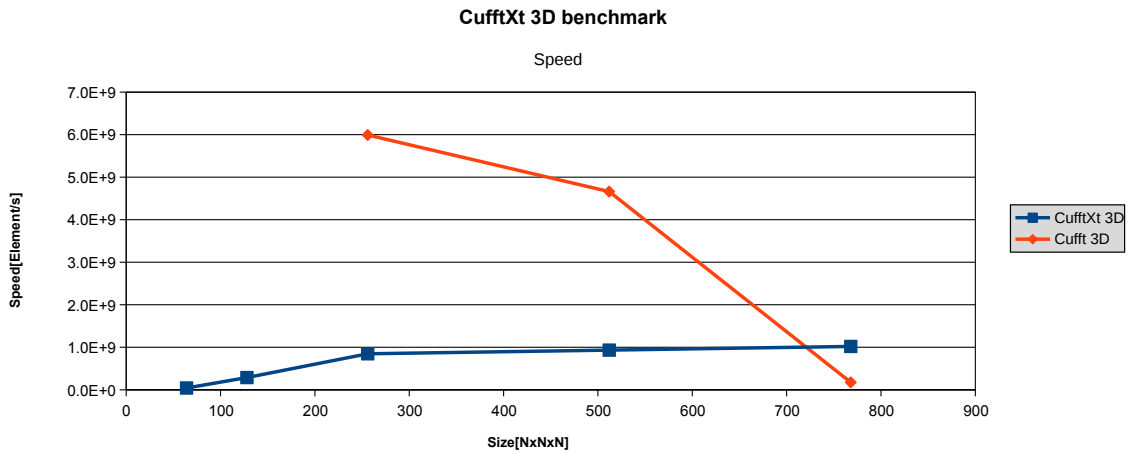
U Barbory je tvar křivky lineární, což může znamenat že rozměr úlohy byl triviální. Do budoucna by bylo vhodné otestovat větší rozměry matic. Do naměřeného času nebyla zahrnuta inicializace a kopírování paměti z CPU na GPU.

Ve srovnání s variantou pro 1 GPU (cuFFT[1]) bylo CufftXt[1] rychlejší pro velké matice kde rychlost cuFFT[1] prudce klesla. Je důležité zmínit že po výpočtu CufftXt[1] jsou data rozložena na jednotlivých kartách. Scatter a gather dat není započítán do výsledného naměřeného času. Aby programy efektivně využily cufftXt[1], měli by nadále pracovat s daty co nejdéle na jednotlivých kartách bez potřeby provádět gather.

4.2.6 CufftXt3D



Obrázek 4.11: Výsledky CufftXt3D na Barboře



Obrázek 4.12: Výsledky CufftXt3D na SC-GPU1

CufftXt3D bylo testováno na velikostech: 64x64x64, 128x128x128, 256x256x256, 512x512x512, 768x768x768.

U Barbory je tvar křivky lineární, což může znamenat že rozměr úlohy byl triviální. Do budoucna by bylo vhodné otestovat větší rozměry matic. Do naměřeného času nebyla zahrnuta inicializace a kopírování paměti z CPU na GPU.

Ve srovnání s variantou pro 1 GPU (cuFFT[1]) bylo cufftXt[1] rychlejší pro velké matice kde rychlost cuFFT[1] prudce klesla. Je důležité zmínit že po výpočtu CufftXt[1] jsou data rozložena na jednotlivých kartách. Scatter a gather dat není započítán do výsledného naměřeného času. Aby programy efektivně využily cufftXt[1], měli by nadále pracovat s daty co nejdéle na jednotlivých kartách bez potřeby provádět gather.

4.3 Závěr předběžného testování

V rámci předběžného testování byly implementovány různé testy, které zkoušely využití unifikované paměti[3][5] pro různé úlohy na více GPU. Dále byla vyzkoušena knihovna cufftXt[1] pro výpočet FFT na více GPU. Testy byly provedeny na 2 přístrojích s odlišným hardware a propojením GPU.

V mnoha testech byly výsledky opačné než se očekávaly, kde Barbora s lepšími GPU a propojením GPU byla pouze o zlomek rychlejší, stejná nebo dokonce pomalejší než SC-GPU1. Pouze u filtrování matice 3x3 filtrem a cufftXt[1] kde se používá nejméně komunikace mezi GPU byla Barbora jednoznačně lepší.

Kapitola 5

Implementace v knihovně k-Wave

Implementace akcelerace pomocí cuFFT[1] a unifikované paměti[3][5] byly prováděny na CUDA variantě knihovny k-Wave[6]. V této kapitole bude krátce popsána struktura knihovny k-Wave[6], zobrazeno chování výpočtu na GPU a ukázány naměřené referenční hodnoty výpočtů na 3D maticích různých rozměrů.

5.1 k-Wave CUDA

Jak již bylo zmíněno knihovna k-Wave[6] je sada nástrojů pro Matlab pro výpočty a simulace ultrazvukových a akustických vln šířících se v tkáních určené především pro výzkumné účely v medicíně.

CUDA implementace se skládá ze spousty modulů pro různé účely jako: čtení/zápis HDF5 souborů, pomocné kernely pro výpočty atd. Z pohledu této práce jsou zajímavé pouze:

- **kSpaceFirstOrderSolver** - jádro výpočtu, zde se nachází hlavní smyčka provádějící iterace simulace
- **Matrix třídy** - kontejnery pro uložení matic obsahující pomocné funkce
- **Pomocné kernely** - obecné často používané výpočty, např. transponování matic

5.1.1 kSpaceFirstOrderSolver

kSpaceFirstOrderSolver obsahuje metodu `compute` která slouží pro inicializaci výpočtu a volání hlavní výpočetní smyčky. Je zde realizována alokace paměti, inicializace plánů pro cuFFT[1], čtení/zápis souborů a definován výpočet.

Z hlavní smyčky je volána varianta metody `computeMainLoop` která realizuje iterativní výpočet. V hlavní smyčce se nacházejí metody: `computeVelocity`, `computeVelocityGradient`, `computeDensityLinear` a další. Tyto metody spočítají příslušné FFT nad určitými maticemi a následně zavolají kernel který s koeficienty FFT pracuje.

5.1.2 Matrix třídy

Implementace k-Wave[6] obsahuje různé třídy pro matice různých typů: `CufftComplexMatrix`, `RealMatrix`, `ComplexMatrix`. Tyto třídy slouží k uchování maticí použitých pro výpočty a

implementuje pomocné funkce jako: kopírování mezi CPU(host) a GPU(device) a případné specifické metody pro daný typ matice.

Cílem akcelerace k-Wave[6] byla třída `CufftComplexMatrix`, která implementuje inicializace plánu cuFFT[1] a samotný výpočet FFT pomocí cuFFT[1]. Konkrétněji je cílem akcelerovat výpočet 3D FFT a to `computeR2CFftND` (dopředné) a `computeC2RFftND`(zpětné).

5.1.3 Pomocné kernely

Zde se nacházejí výhradně kernely pro transponování 3D matic podle os X a Z. Kernely transponování používají stejnou implementaci která byla zmíněna v podkapitole 3.4 Transponování 2D matic. Transponování bude důležité pro implementaci FFT využívající více GPU bez použití cuFFTXt[1], které nese jisté omezení.

5.2 Model výpočtu

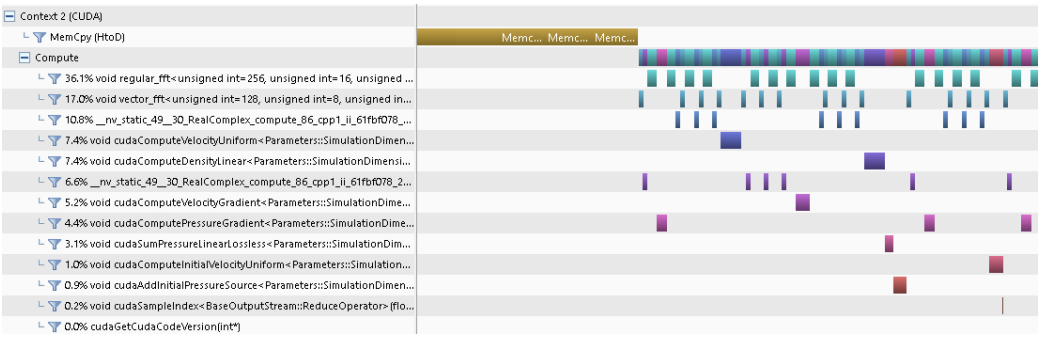
Pro rozpoznání kritických oblastí vhodných pro akceleraci bylo provedeno profilování aplikace. Pro testovací účely se solver spouštěl s následujícími parametry příkazem:

```
./kspaceFirstOrder-CUDA -i ../inputs/256x256x256_in_add.h5 -o /
../outputs/reference.h5 --benchmark 500
```

kde:

- `-i` - vstupní HDF5 soubor nesoucí počáteční nastavení simulace a její parametry
- `-o` - cesta pro HDF5 soubor obsahující výsledek simulace
- `--benchmark` - parametr kterým se nastavuje pevně počet iterací

Pro zobrazení průběhu výpočtu byl program profilován příkazem `nvprof`. Výsledek je zobrazen pomocí programu Nvidia Visual Profiler.



Obrázek 5.1: Výstup nvprof pro výpočet 256x256x256 matice v původní implementaci zobrazený v Nvidia Visual Profileru.

Na časovém průběhu volaní kernelů lze vidět část inicializace před výpočtem (kopírování dat na GPU) a první iteraci výpočtu. Je důležité zmínit, že program po inicializaci pracuje na GPU a nekopíruje data zpět na CPU. Až při uložení výsledků (po skončení výpočtu) jsou data zkopírována z GPU na CPU.

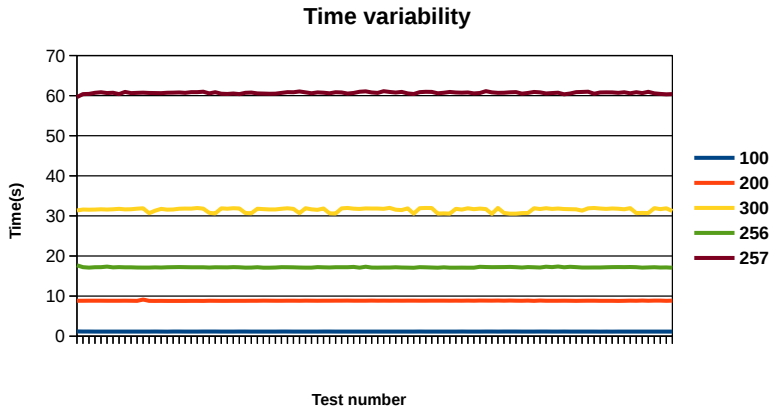
Největší část výpočtů na GPU tvoří výpočet FFT. Konkrétně se jedná o 70.5% výpočtu. Zbytek tvoří kernely pracující s koeficienty z FFT. Je tedy zřejmé, že pro urychlení výpočtu je vhodné se zaměřit na výpočet FFT.

5.3 Měření referenčních hodnot

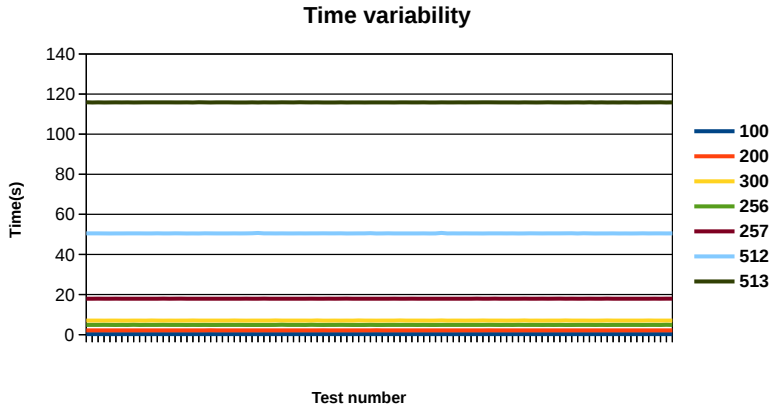
Pro vyhodnocení zrychlení je potřeba naměřit referenční hodnoty pro rychlost původní implementace. Zároveň bylo cílem tohoto měření zjistit potřebný počet naměřených hodnot pro pravdivé výsledky, vlivem možného běhu ostatním programů na pozadí během výpočtu.

Pro měření byly použity vstupy se stejnými simulačními parametry a rozměry: 100^3, 200^3, 256^3, 257^3, 300^3, 512^3 a 513^3. Všechny velikosti byly simulovány pro 500 kroků. Pro každou velikost vstupu bylo naměřeno 100 hodnot na obou strojích SC-GPU1 a Barbora.

V následujících grafech je ukázán rozptyl naměřených hodnot.



Obrázek 5.2: Naměřené časy pro jednotlivé výpočty na stroji SC-GPU1



Obrázek 5.3: Naměřené časy pro jednotlivé výpočty na stroji Barbora

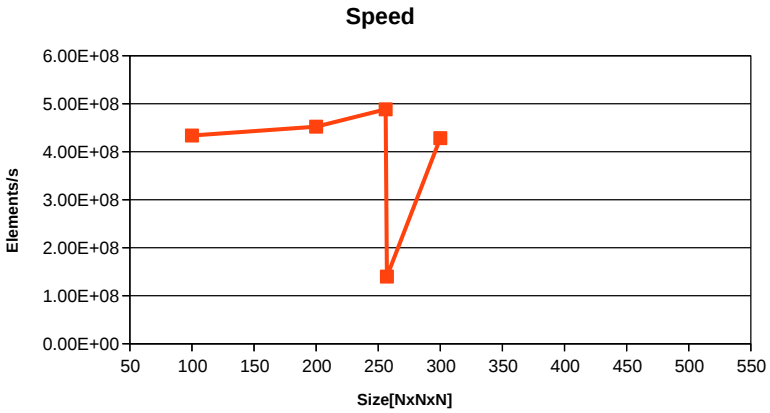
Z naměřených hodnot lze pozorovat, že na SC-GPU1 nastává menší variabilita mezi naměřenými hodnotami, jelikož je veřejně přístupný a může na něm běžet více úloh najednou. Největší směrodatná odchylka pro naměřené časy se vyskytovala u případu 300^3 a byla rovna 0.48 sekundám, což ve srovnání s průměrnou dobou trvání, které byla rovna 31.5 sekundám, není příliš zásadní.

Na Barboře byly naměřené hodnoty velice konzistentní, jelikož při výpočtu je celý uzel rezervován. Největší směrodatná odchylka pro naměřené časy se vyskytovala u případu 512^3

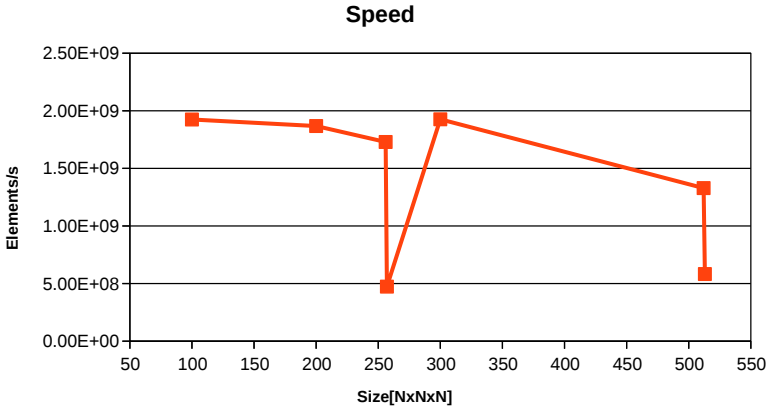
a byla rovna 0.039 sekundám, což ve srovnání s průměrnou dobou trvání, které byla rovna 50.5 sekundám, není vůbec zásadní.

Z výsledků bylo rozhodnuto, že pro budoucí testy bude provedeno 10 měření pro testy trvající méně než minutu. Pro ostatní bude počet měření zvolen podle doby trvání.

V následujících grafech je ukázána rychlost pomocí počtu zpracovaných prvků matice za sekundu.



Obrázek 5.4: Naměřená rychlost pro jednotlivé velikosti na stroji SC-GPU1



Obrázek 5.5: Naměřená rychlost pro jednotlivé velikosti na stroji Barbora

Na stroji SC-GPU1 nejsou naměřena data pro velikosti 512^3 a 513^3, protože grafické karty mají příliš málo paměti VRAM a program skončí neúspěšně. Z grafů lze pozorovat, že pro velikosti 257^3 a 513^3 je program výrazně pomalejší. Důvodem je špatně zarovnaná paměť, která vede k nezarovnanému přístupu do paměti který vede k velké ztrátě výkonu.

Obecně stroj Barbora je zhruba 4x rychlejší než SC-GPU1, což by se dalo očekávat vzhledem k tomu, že Barbora má mnohem výkonnější GPU s novější architekturou.

Kapitola 6

Převod na unifikovanou paměť

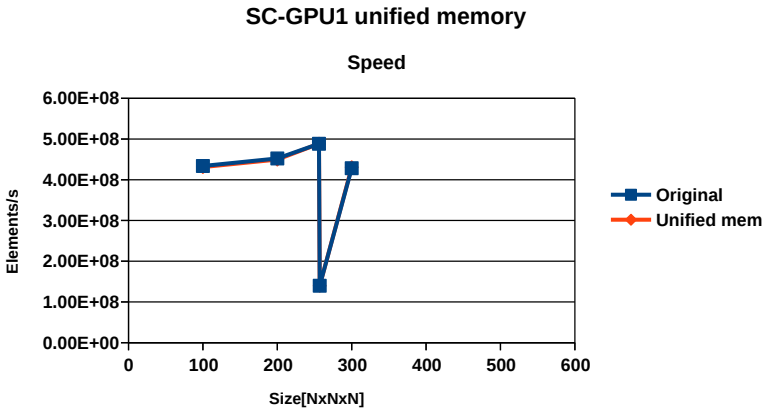
V této části implementace byl převeden původní paměťový model, kde je odděleně alokovaná paměť na CPU a GPU, na unifikovanou paměť[3][5]. Pro převod na unifikovanou paměť[3][5] stačilo upravit třídy pro uložení matic, jelikož přístupy na ukazatele a jejich manipulace byly zapouzdřeny v samotné třídě.

Původně byly používány dva ukazatele pro ukládání dat `mDeviceData` a `mHostData`, kde `mDeviceData` byl ukazatel na paměť alokovanou na GPU a `mHostData` byl ukazatel na paměť alokovanou na CPU.

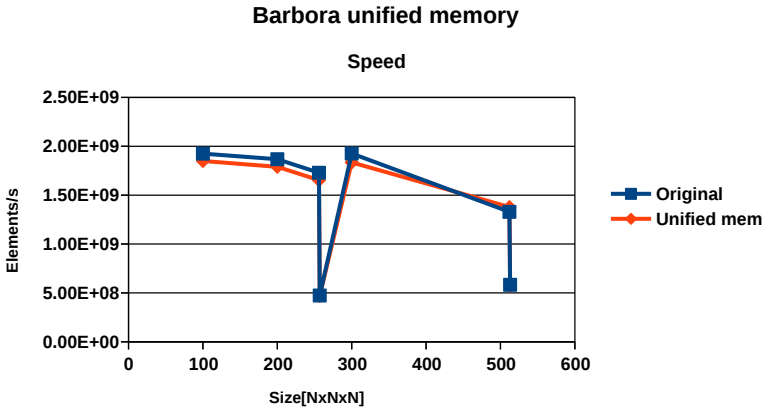
Pro převod na unifikovanou paměť[3][5] byly ukazatelé nahrazeny jediným ukazatelem `mData`. Při alokaci stačilo alokovat pouze jeden blok paměti pomocí `cudaMallocManaged`. Ostatní moduly přistupovali do paměti přes metody `getHostData` a `getDeviceData` a explicitně museli volat kopírování z GPU na CPU a obráceně. Pro jednoduchost implementace `getHostData` a `getDeviceData` vracejí ukazatel `mData` a metody pro kopírování dat mezi CPU a GPU byly zrušeny díky vlastnostem unifikované paměti[3][5]. Přímé přístupy do pole z CPU nebo GPU nemuseli být ošetřeny díky vlastnostem unifikované paměti[3][5].

6.1 Testování unifikované paměti

Měření bylo provedeno pro stejné parametry které byly popsány v předchozí kapitole. Výsledky byly srovnány s původní implementací. Výpočet je stále prováděn na jedné kartě. Níže jsou uvedeny grafy výsledných rychlostí.



Obrázek 6.1: Naměřená rychlost pro jednotlivé velikosti na stroji SC-GPU1

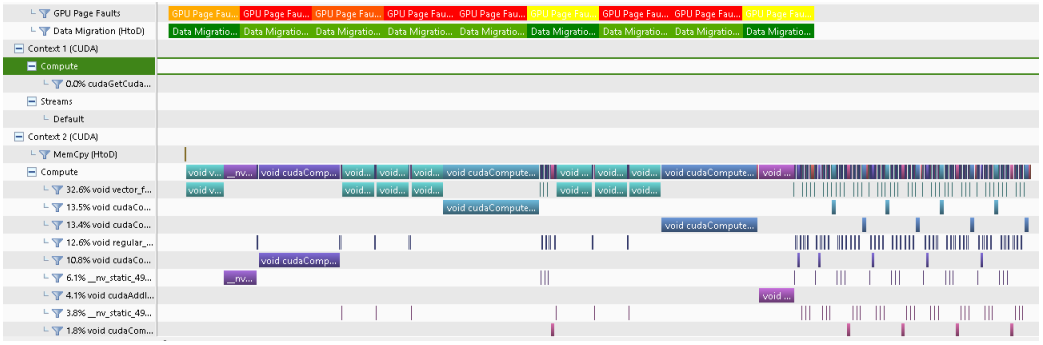


Obrázek 6.2: Naměřená rychlost pro jednotlivé velikosti na stroji Barbora

Z grafů lze vidět, že nedošlo ke snížení ani k zvýšení rychlosti. I přesto, že nebyl implementován žádný "prefetch", který by dopředně zkopíroval chybějící stránky na GPU.

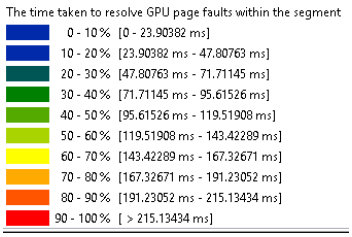
Na SC-GPU1 výpočet pro matice o rozměrech 512^3 a 513^3 byl možný díky vlastnostem unifikované paměti[3][5], ale nebyl zahrnut do výsledků. Proč výsledky nebyly zahrnuty je vysvětleno v další podkapitole.

Dále bude ukázán průběh výpočtu a chování unifikované paměti[3][5] na následujících snímcích.



Obrázek 6.3: Výtup nvprof pro výpočet 256x256x256 matice používající unifikovanou paměť zobrazený v Nvidia Visual Profileru.

Na časovém průběhu volaní kernelů lze nahoře vidět komunikaci mezi CPU a GPU `Data Migration (HtoD)` a naměřené úseky výpadků stránek `GPU Page Faults` v rámci unifikované paměti[3][5]. Barvy těchto úseků jsou odvozeny od škály vytvořené profilovacím nástrojem a jsou spíše orientační.



Obrázek 6.4: Ukázka škály pro úseky výpadků stránek pro unifikovanou paměť

Na začátku se veškerá data unifikované paměti[3][5] vyskytují na CPU. Při první iteraci se k těmto datům poprvé přistupuje na GPU, což vyvolává výpadky stránek a tak i kopírování dat z CPU na GPU. Z obrázků profilování to lze jednoduše vidět na spojitém úseku `Data Migration (HtoD)` a `GPU Page Faults` a dlouhého trvání kernelů první iterace oproti zbytku výpočtu.

Po první iteraci jsou veškerá data na GPU. Z obrázku lze vidět, že nedochází k žádným výpadkům stránek ani komunikaci mezi CPU a GPU a výpočet kernelů je mnohem rychlejší.

```
==689421== Unified Memory profiling result:
Device "GeForce GTX 1080 (0)"
   Count  Avg Size  Min Size  Max Size  Total Size  Total Time  Name
   33710  34.076KB  4.0000KB  0.9883MB  1.095520GB  186.5496ms  Host To Device
    2234         -         -         -           -  396.4077ms  Gpu page fault groups
Total CPU Page faults: 3839
```

Obrázek 6.5: Textový výpis profilování pro unifikovanou paměť 256x256x256 matice

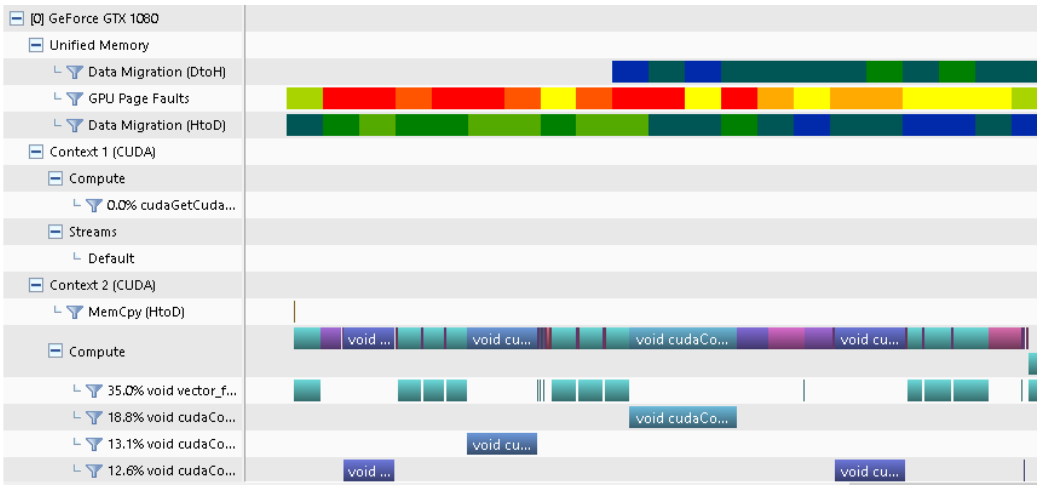
Z výstupu `nvprof` došlo k přesunu 1GB paměti, což odpovídá velikosti paměti použité na GPU v původní implementaci. Výpadky stránek zpomalují první iteraci simulace, ale na druhou stranu není potřeba explicitně kopírovat data z CPU na GPU. Kopírování dat se děje za běhu první iterace.

6.2 Speciální vlastnost unifikované paměti

V původní implementaci na stroji SC-GPU1 se nepodařilo program spustit pro matice velikosti 512^3 a 513^3, jelikož nestačila paměť na grafické kartě. S unifikovanou pamětí[3][5] je možné automaticky mít pouze potřebné stránky paměti na GPU.

Při alokaci unifikované paměti[3][5] není zabrán prostor na kartě, pouze na straně CPU. Paměť na GPU je vyčerpávána pouze při přístupech do unifikované paměti[3][5] ze strany GPU a následné poskytnutí příslušných paměťových stránek z CPU a jejich uložení na GPU.

Pokud na GPU není místo pro další stránky, GPU začne předchozí stránky zahazovat a ukládat je zpět na CPU ("swapování"). V následujícím výstupu `nvprof` je ukázána situace kdy dochází ke "swapovaní" na matici o velikost 512^3.



Obrázek 6.6: Výstup nvprof pro výpočet 512x512x512 matice používající unifikovanou paměť zobrazený v Nvidia Visual Profileru.

Z výstupu na obrázku lze vidět, že po určité době začne docházet ke komunikaci GPU směrem k CPU (`DtoH`), kde dochází k odkládání stránek. Bohužel tenhle jev vede k velkému zpomalení programu.

```
==689663== Unified Memory profiling result:
Device "GeForce GTX 1080 (0)"
   Count  Avg Size  Min Size  Max Size  Total Size  Total Time  Name
 1295104  33.856KB  4.0000KB  0.9961MB  41.81726GB   6.161499s  Host To Device
   17946  1.9991MB  64.000KB  2.0000MB  35.03497GB   4.678731s  Device To Host
   77074         -         -         -           -  18.836928s  Gpu page fault groups
Total CPU Page faults: 27356
```

Obrázek 6.7: Textový výpis profilování pro unifikovanou paměť 512x512x512 matice pro 5 iterací

Z textového výpisu `nvprof` lze vidět obrovský objem přesunutých dat. Je nutné podotknout, že tenhle test byl nastaven pouze na 5 iterací oproti 500 iteracím na kterých se ostatní testy provádějí.

Kapitola 7

Multi-GPU implementace

7.1 CuFFT implementace

7.1.1 Implementace

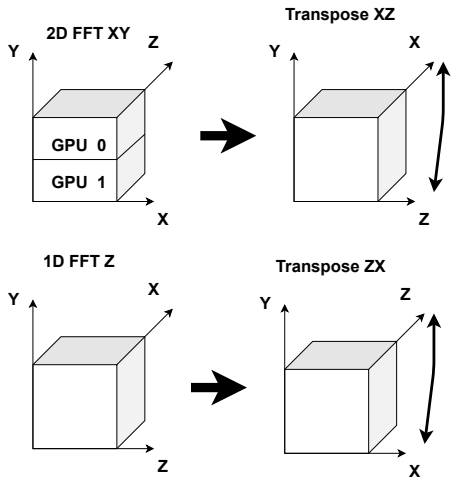
Pro implementaci 3D FFT pomocí více GPU používající cuFFT[1] byl zvolen postup podobný postupu popsaného v práci pana Jiřího Jaroše[4].

V první fázi je spočteno 2D FFT v osách XY pro všechny roviny v ose Z. Tento výpočet je rozdělen mezi všechna GPU, kde každé GPU vypočítá několik souvislých řezů.

V druhé fázi je matice transponována v osách X a Z. Transponování je provedeno na jednom GPU, jelikož předchozí měření transponování na více GPU vedlo k slabým výsledkům. Matice je transponována, aby data pro další fázi byla vhodně uspořádána pro výpočet.

Ve třetí fázi je spočteno 1D FFT v ose X (osa Z před transponováním matice). Výpočet této fáze probíhá na jednom GPU, jelikož se v druhé fázi počítá transponování na jednom GPU, což znamená že data jsou uložena na stejném GPU. Při počítání na více GPU by driver zpětně kopíroval data mezi GPU, což by vedlo k zpomalení.

V poslední fázi je matice zpětně transponovaná v osách X a Z.



Obrázek 7.1: Schéma výpočtu 3D FFT pro více GPU podle výše uvedeného popisu

Samotná implementace probíhala v rámci třídy `CufftComplexMatrix`. Zde bylo potřeba vytvořit několik cuFFT[1] plánu pro každé GPU pro každou variantu FFT. Celkem bylo potřeba implementovat varianty `R2CFftPlanND` a `C2RFftPlanND`.

Před samotnou simulací je potřeba inicializovat cuFFT[1] plány které slouží pro popis jednotlivých výpočtů cuFFT[1]. Pro 1. fázi výpočtu má každé GPU vytvořeno plán pomocí `cufftPlanMany`, který definuje výpočet několika 2D FFT v ose Z. Každé GPU počítá celkem rozměr Z děleno počtem GPU 2D desek. Pokud rozměr Z není dělitelný počtem GPU, je poslednímu GPU přidělen zbytek po dělení 2D desek.

Pro 3. fázi je vytvořen plán pro jedno zvolené GPU pomocí `cufftPlanMany`. Tento plán provede $X * Y$ 1D FFT výpočtů.

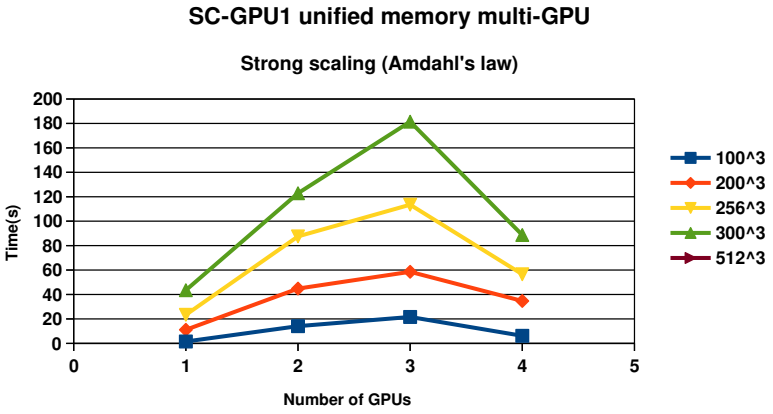
V samotných metodách pro výpočet 3D FFT je implementace totožná s výše uvedeným postupem. V první fázi jsou na každém GPU spočítány 2D FFT v ose Z. Následně je matice transponována podle os X a Z na jednom GPU. Poté jsou nad touto maticí spočítány 1D FFT a nakonec je matice zpětně transponována podle os X a Z.

7.1.2 Měření

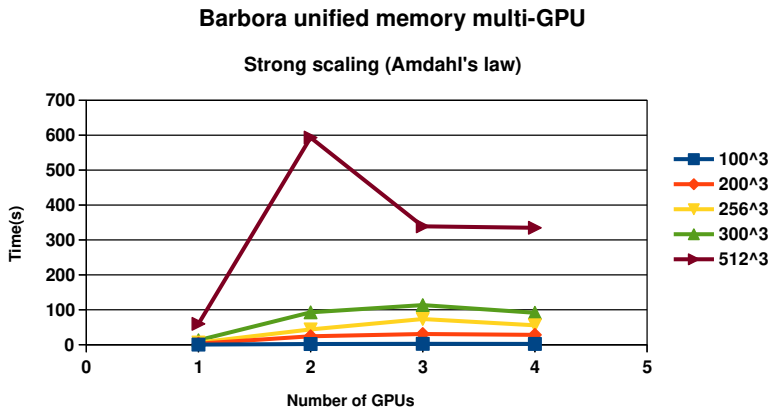
Měření bylo provedeno pro velikosti matic: 100^3, 200^3, 256^3, 300^3 a 512^3. Počet iterací byl nastaven na 500 pro všechny velikosti matic. Z měření byly sestaveny grafy pro silné škálování a slabé škálování.

Pro silné škálovaní byly použity velikosti matic: 100^3, 200^3, 256^3, 300^3, 512^3. Graf silného škálování ukazuje jak moc pro úlohu stejného rozměru přidávání výpočetních jednotek urychluje výpočet.

Pro slabé škálovaní byly použity velikosti matic: $64\times64\times N$, $128\times128\times N$ a $256\times256\times N$. N se s rostoucím počtem GPU zvětšuje, tak aby poměr rozměru úlohy na počet GPU zůstal stejný. Konkrétně se N spočítalo jako rozměr osy X krát počet GPU. Graf slabého škálování ukazuje jak moc je možné škálovat rozměry úlohy s počtem výpočetních prostředků při ideálně stejném výpočetním čase.



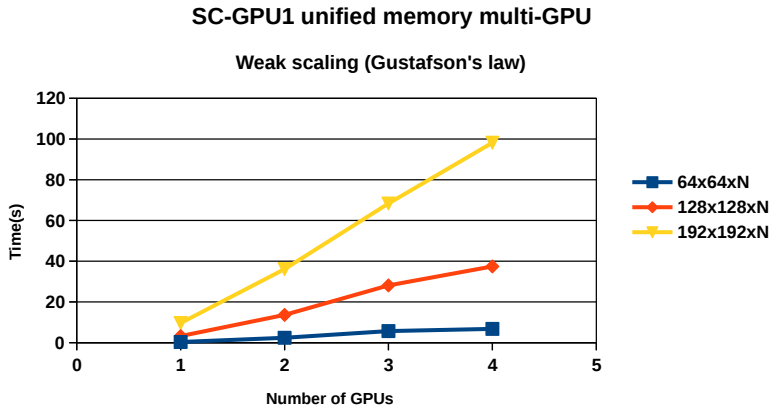
Obrázek 7.2: Graf silného škálování na stroji SC-GPU1



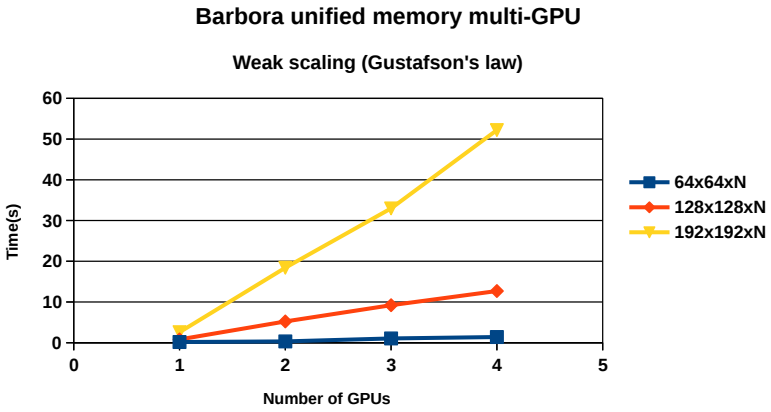
Obrázek 7.3: Graf silného škálování na stroji Barbora

Z výsledků silného škálování lze vidět, že při použití více jak jednoho GPU se doba výpočtu zpomaluje. Pro jedno GPU je implementace v průměru pro všechny velikosti 0.8krát pomalejší než originální implementace. V ostatních případech je 10krát pomalejší.

Na SC-GPU1 implementace pro 4 GPU škáluje správným směrem dolů oproti ostatním. Na Barboře tomu bylo podobně s tím, že u velikosti $512 \times 512 \times 512$ byl náhlý vzrůst výpočetního času pro 2 GPU. Je možné že od dvou GPU dochází ke komunikaci mezi GPU přes jednotlivé CPU sokety nebo GPU navzájem přistupují na stejné paměťové stránky a tak dochází k tzv. “memory trashing”.



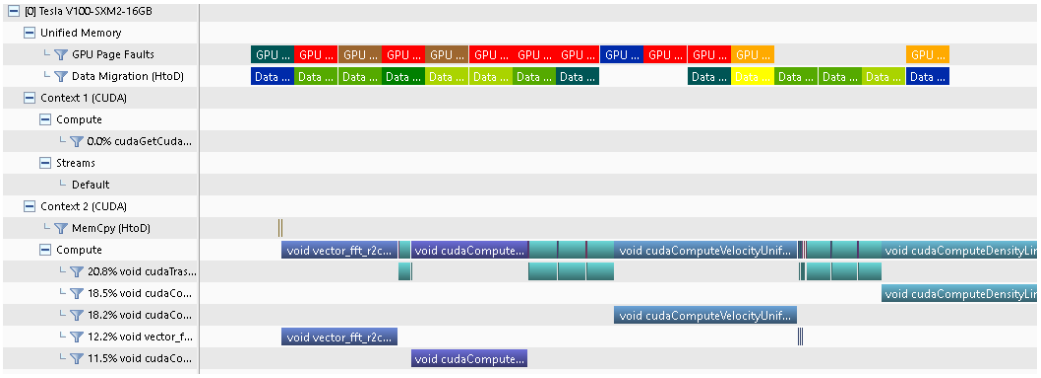
Obrázek 7.4: Graf slabého škálování na stroji SC-GPU1



Obrázek 7.5: Graf slabého škálování na stroji Barbora

Z výsledků slabého škálování lze vidět, že při použití více GPU na zvyšující se rozměry úlohy se doba výpočtu zvyšuje.

Výsledky této metody pro více GPU nedopadli dobře. Z nějakého důvodu při použití více GPU se doba výpočtu zhoršuje ve srovnání s původní implementací. Pro identifikaci problému byl program profilován při výpočtu s jedním a se dvěma GPU.



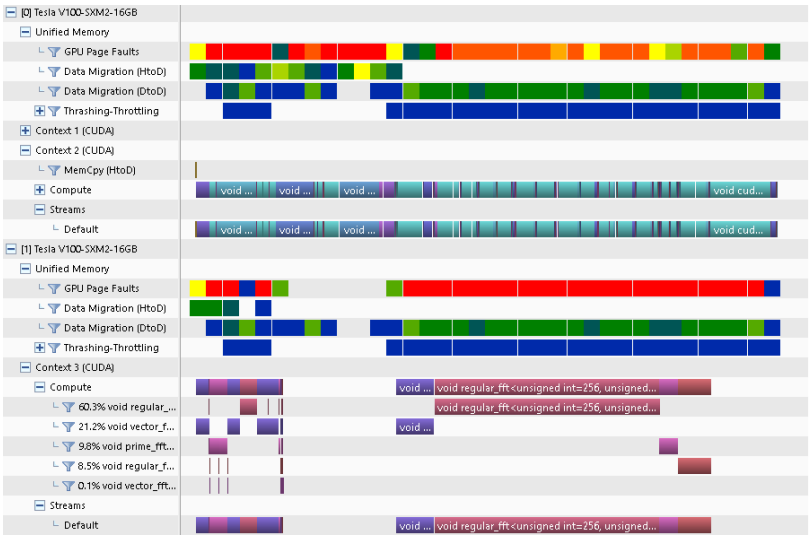
Obrázek 7.6: Výstup nvprof pro výpočet 256x256x256 matice na jediném GPU zobrazený v Nvidia Visual Profileru.

```
==951749== Unified Memory profiling result:
Device "GeForce GTX 1080 (0)"
   Count  Avg Size  Min Size  Max Size  Total Size  Total Time  Name
   30550  37.602KB  4.0000KB  0.9961MB  1.095520GB  187.1557ms  Host To Device
    2237         -         -         -           -  415.9441ms  Gpu page fault groups
Total CPU Page faults: 3842
```

Obrázek 7.7: Textový výpis profilování pro unifikovanou paměť 256x256x256 matice na jediném GPU pro 500 iterací

Z výpisu profilování a výstupu Nvidia Visual Profileru má program podobné chování jako původní implementace s paměťovým modelem převedeným na unifikovanou paměť[3][5]. Samotný výpis pro unifikovanou paměť[3][5] má skoro stejné chování. Jediným důvodem

proč je tato implementace pomalejší je režie transponování apod. Transponování zabíralo 20% výpočetního času na GPU.



Obrázek 7.8: Výstup nvprof pro výpočet 256x256x256 matice na dvou GPU zobrazený v Nvidia Visual Profileru.

```
==953015== Unified Memory profiling result:
Device "GeForce GTX 1080 (0)"
   Count  Avg Size  Min Size  Max Size  Total Size  Total Time  Name
   26927  38.269KB  4.0000KB  0.9844MB  0.982727GB  164.5039ms  Host To Device
  263018         -         -         -           -  37.410719s  Gpu page fault groups
   45943         -         -         -           -  24.795544s  Page throttles
 1956574  4.0000KB  4.0000KB  4.0000KB  7.463738GB           -  Memory thrashes
    5652  4.0000KB  4.0000KB  4.0000KB  22.07813MB           -  Remote mapping from device
    2504  4.0000KB  4.0000KB  4.0000KB  9.781250MB           -  Remote mapping to device
 4293880  23.870KB  4.0000KB  0.9961MB  97.74964GB  14.729266s  Transfers from Device
 4418850  23.241KB  4.0000KB  0.9961MB  97.94556GB  13.938095s  Transfers to Device
Total CPU Page faults: 3844
```

Obrázek 7.9: Textový výpis profilování GPU 0 pro unifikovanou paměť 256x256x256 matice na dvou GPU pro 500 iterací

Z výpisu profilování a výstupu Nvidia Visual Profileru lze vidět že dochází k nadměrné komunikaci mezi GPU. K nadměrné komunikaci mezi GPU dochází, jelikož každé GPU v první fází algoritmu přistupuje k části matice která po skončení výpočtu 3D FFT leží vždy na prvním GPU, takže pro každý výpočet FFT dojde vždy k přenosu dat mezi GPU. Transponování matic zabírá 60% výpočetního času.

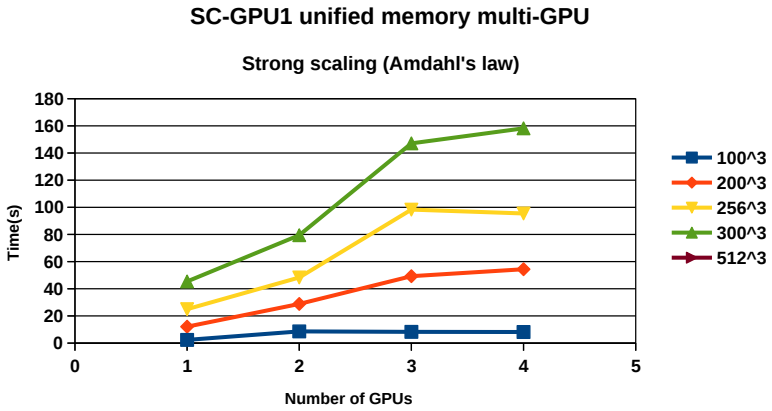
7.2 CuFFT implementace používající prefetch

7.2.1 Implementace

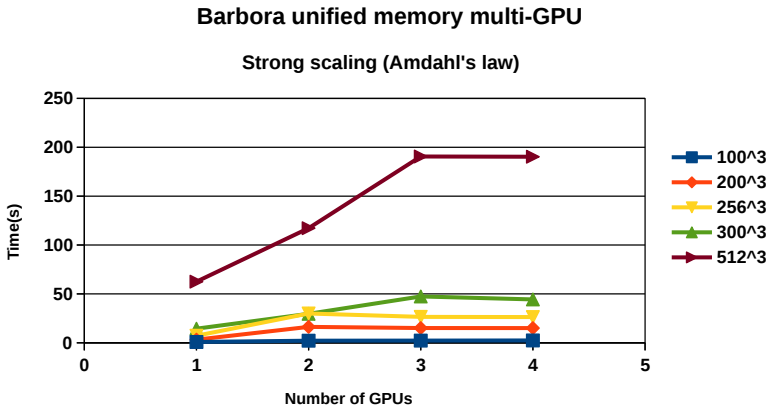
Jedná se v podstatě o stejnou implementaci jako předchozí, s tím rozdílem, že před první fází jsou data na jednotlivá GPU přednačtena pomocí `cudaMemPrefetchAsync`. Cílem bylo omezit memory thrashing a tím zlepšit výkon.

7.2.2 Měření

Měření bylo provedeno pro stejné parametry jako v předchozí kapitole 7.1



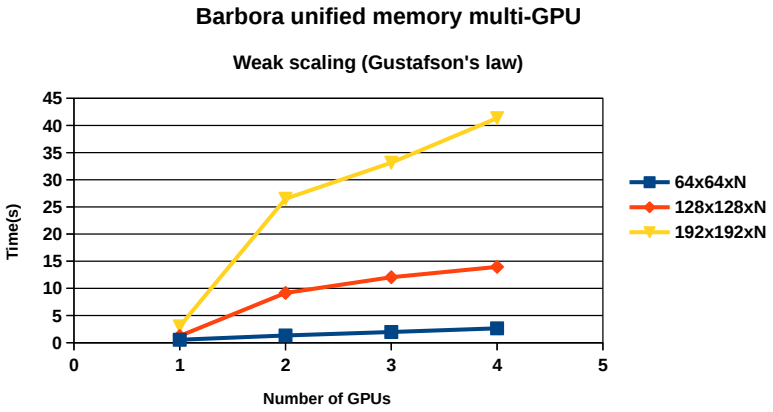
Obrázek 7.10: Graf silného škálování na stroji SC-GPU1



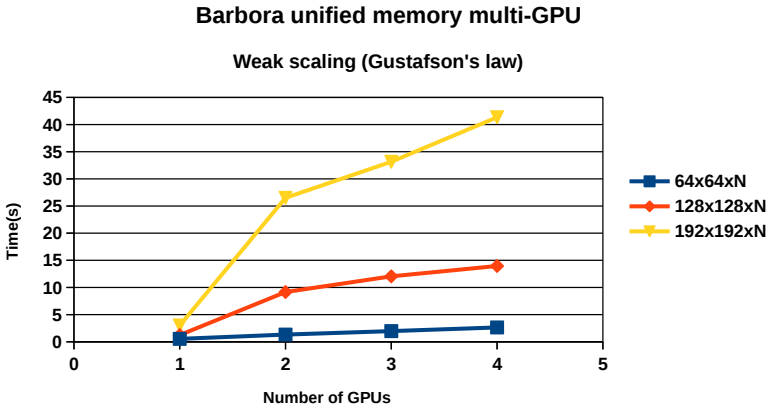
Obrázek 7.11: Graf silného škálování na stroji Barbora

Z výsledků silného škálování lze vidět, že při použití více jak jednoho GPU se doba výpočtu zpomaluje. Na rozdíl od předchozí implementace se výkon pro jedno GPU zhoršil a výkon pro více GPU zlepšil. Tato implementace je ale stále pomalejší než původní implementace.

Na obou strojích se pro 4 použité GPU začíná výpočet škálovat správným směrem. Obecně pro více GPU měl "prefetch" pozitivní dopad.



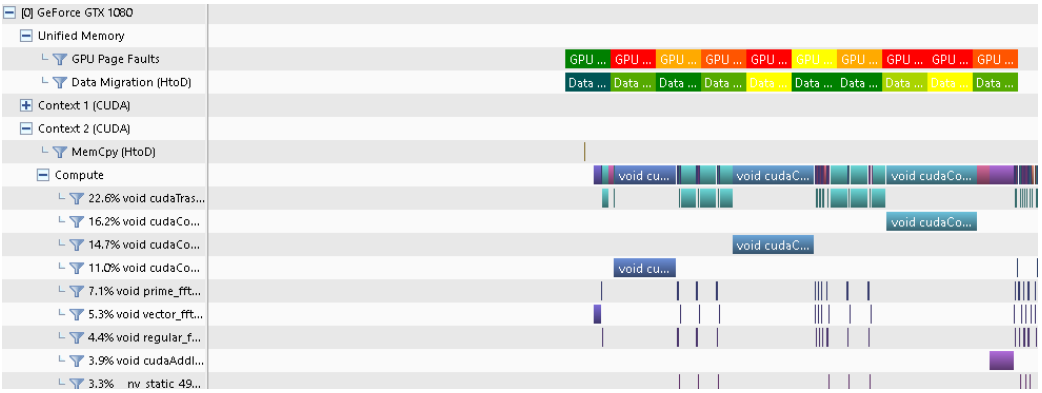
Obrázek 7.12: Graf slabého škálování na stroji SC-GPU1



Obrázek 7.13: Graf slabého škálování na stroji Barbora

Z výsledků slabého škálování lze vidět, že při použití více GPU na zvyšující se rozměry úlohy se doba výpočtu zvyšuje. Doba výpočtu se zvětšuje menším tempem, než v předešlé implementaci bez “prefetch”.

Výsledky této metody pro více GPU nedopadli dobře. Doby výpočtů se pro více GPU zmenšily, ale oproti původní implementaci je tahle implementaci pořád pomalejší. Pro pozorování dopadu “prefetchingu” na běh výpočtu byl program profilován při výpočtu s jedním a se dvěma GPU.

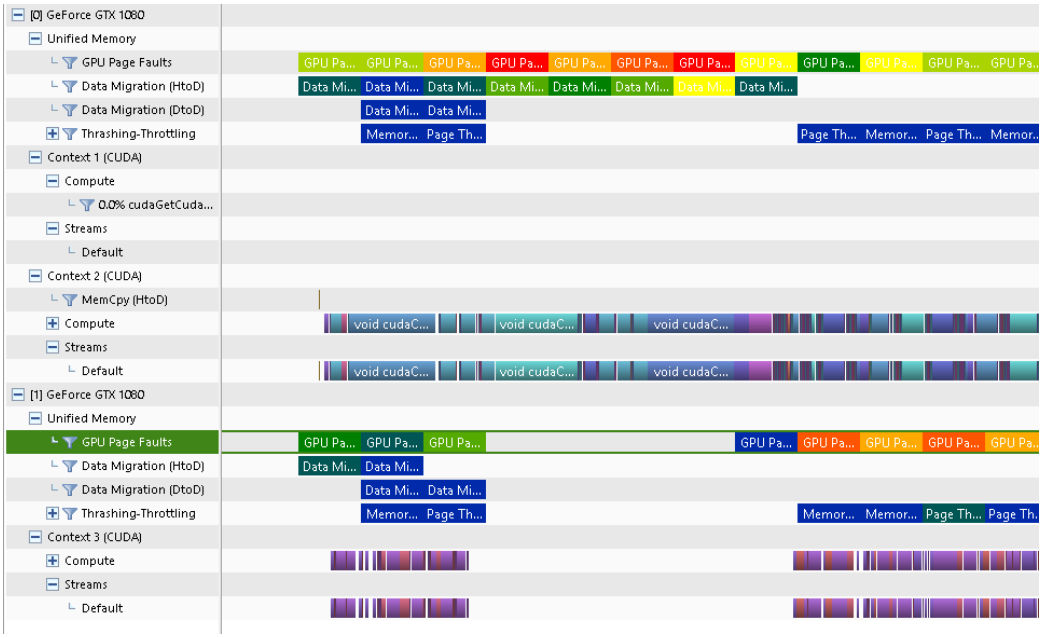


Obrázek 7.14: Výstup nvprof pro výpočet 256x256x256 matice na jediném GPU zobrazený v Nvidia Visual Profileru.

```
==1224700== Unified Memory profiling result:
Device "GeForce GTX 1080 (0)"
   Count  Avg Size  Min Size  Max Size  Total Size  Total Time  Name
   28721  39.996KB  4.0000KB  2.0000MB  1.095520GB  200.9130ms  Host To Device
    2080         -         -         -           -  379.3598ms  Gpu page fault groups
Total CPU Page faults: 3838
```

Obrázek 7.15: Textový výpis profilování pro unifikovanou paměť 256x256x256 matice na jediném GPU pro 500 iterací

I přesto, že výpočet byl pomalejší s "prefetchem" jsou výstupy `nvprof` a Nvidia Visual Profileru téměř stejné jako u předchozí implementace. Je možné, že `cudaMemPrefetchAsync` zavádí určitou režii která zpomaluje výpočet.



Obrázek 7.16: Výstup nvprof pro výpočet 256x256x256 matice na dvou GPU zobrazený v Nvidia Visual Profileru.

```
Device "GeForce GTX 1080 (0)"
   Count  Avg Size  Min Size  Max Size  Total Size  Total Time  Name
   25167  40.955KB  4.0000KB  2.0000MB  0.982971GB  180.0730ms  Host To Device
  226629         -         -         -           -  33.193659s  Gpu page fault groups
   36014         -         -         -           -  18.850190s  Page throttles
 1397846  4.0000KB  4.0000KB  4.0000KB  5.332359GB           -  Memory thrashes
    2438  4.0000KB  4.0000KB  4.0000KB  9.523438MB           -  Remote mapping from device
    1551  4.0000KB  4.0000KB  4.0000KB  6.058594MB           -  Remote mapping to device
 3093441  35.899KB  4.0000KB  2.0000MB  105.9082GB  14.587306s  Transfers from Device
 3573290  31.158KB  4.0000KB  2.0000MB  106.1805GB  14.342346s  Transfers to Device
```

Obrázek 7.17: Textový výpis profilování pro unifikovanou paměť 256x256x256 matice na dvou GPU pro 500 iterací

Z výstupu Nvidia Visual Profileru lze vidět že dochází k menším výpadkům stránek a obecně menší komunikaci na druhém GPU. V předchozí implementaci zabíralo spoustu času transponování matic a docházelo často k "memory trashingu". V současné implementaci se z grafického výstupu profileru zdá, že počet výpadků stránek a "memory trashing" byl snížen, ovšem textový výpis `nvprof` uvádí, že se objem komunikace zvýšil.

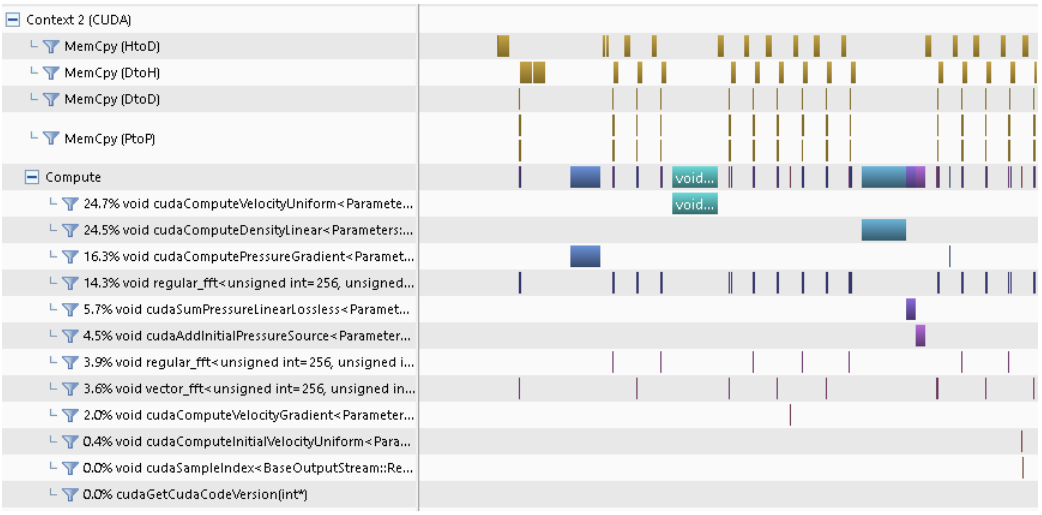
7.3 CuFFTXt

7.3.1 Implementace

Jako poslední implementace pro akceleraci byla zvolena varianta cuFFTXt[1]. I přestože je snahou se vyhnout implementaci pomocí cuFFTXt[1] kvůli jejím omezením, je vhodné uvést výsledky knihovny navržené pro více GPU oproti vlastní implementaci.

Implementace pomocí cuFFTXt[1] byla implementována pouze orientačně bez ohledu na správnosti výsledku pro změření výkonu. Pro úplnou a efektivní implementaci by bylo potřeba upravit ostatní části knihovny, aby využívali více GPU, což bohužel z časových důvodů nebylo možné. Testy budou dostatečně přesně simulovat běh výpočtu pro přesné odhadnutí výkonu správné implementace.

Důvodem je, že cuFFTXt[1] má vstupy a výstupy uložené na všech GPU zvlášť, takže pro kernely pracujících na jednom GPU by se museli data pomocí gather poslat na dané GPU a pak po dokončení výpočtu zpětně poslat pomocí scatter na ostatní GPU, což by bylo velice pomalé. Na dolním obrázku je taková implementace ukázána.



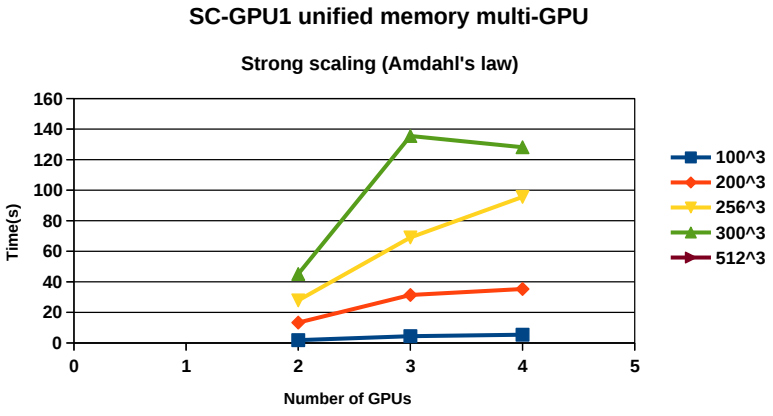
Obrázek 7.18: Výstup nvprof pro výpočet 256x256x256 matice na dvou GPU pomocí neefektivní implementace zobrazený v Nvidia Visual Profileru.

Na horním obrázku je uvedeno profilování nevhodné implementace, kde před každým voláním plánu cuFFTXt[1] jsou data pomocí `cufftXtMemcpy` zkopírována na všechny GPU a po dokončení pomocí `cufftXtMemcpy` zkopírována zpět ze všech GPU. Je vidět že mezi modrými úseky realizující výpočet je spousta komunikace mezi GPU a CPU. Celkově je implementace velice pomalá.

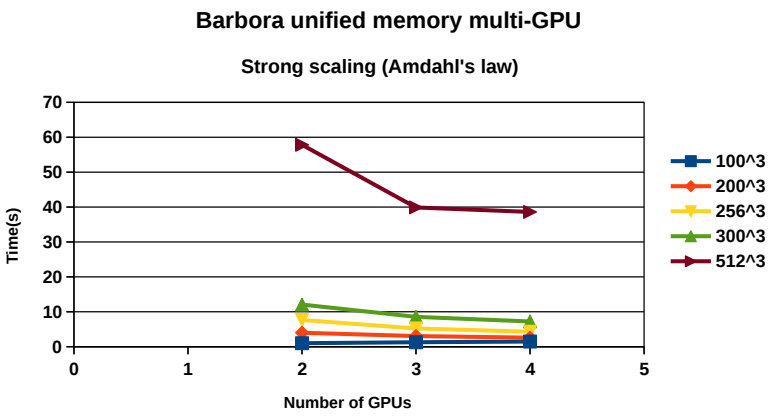
CuFFTXt[1] implementace byla realizována jako výpočet cuFFTXt[1] nad neinicializovaným polem, které se dále ve výpočtu nepoužívalo.

7.3.2 Měření

Měření bylo provedeno pro stejné parametry jako v předchozí podkapitole 7.1. Měření nebylo možno provést pro jedno GPU kvůli omezení cuFFTXt[1].



Obrázek 7.19: Graf silného škálování na stroji SC-GPU1

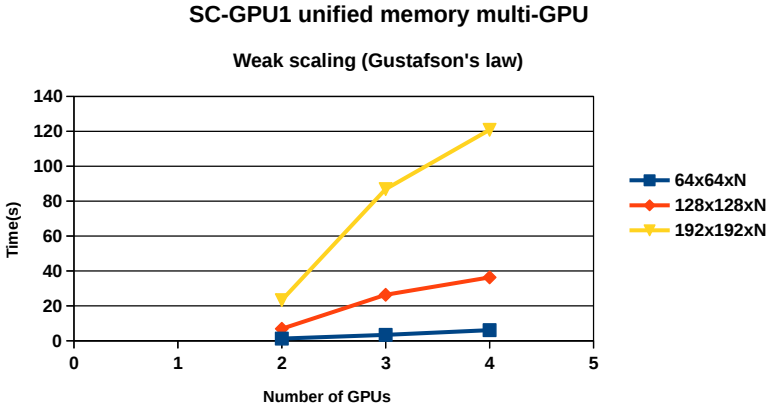


Obrázek 7.20: Graf silného škálování na stroji Barbora

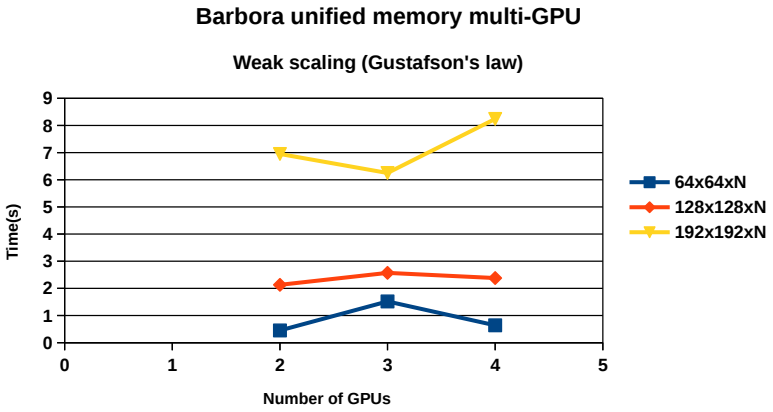
Z výsledků silného škálování lze vidět, že při použití více jak jednoho GPU se doba výpočtu na SC-GPU1 zpomaluje, kdežto na Barboře se zrychluje. Důvodem je že GPU na SC-GPU1 jsou propojeny pomocí PCI-e a GPU na Barboře jsou propojeny pomocí NVLink[2].

Na Barboře dobře škáluje úloha o rozměrech 512^3, ostatní úlohy jsou pouze zlomkově rychlejší nebo v případě 100^3 pomalejší. Faktorem bude propustnost a latence propojení GPU. Pouze matice o rozměrech 512^3 pro 3 a více GPU byla o 20-30% rychlejší než původní implementace. Pro ostatní rozměry a počty GPU byla implementace pomalejší než původní.

Pro SC-GPU1 byly všechny výsledky pomalejší než původní implementace.

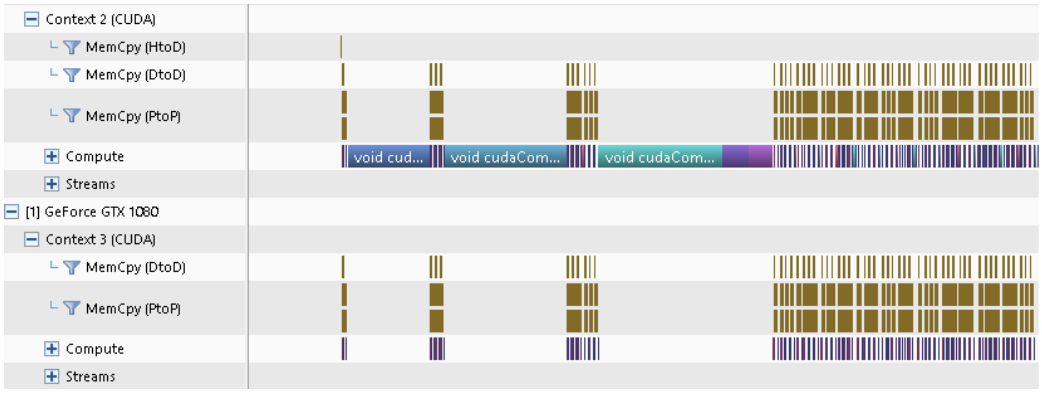


Obrázek 7.21: Graf silného škálování na stroji SC-GPU1



Obrázek 7.22: Graf silného škálování na stroji Barbora

Z výsledků slabého škálování lze vidět, že pro Barboru zůstává doba výpočtu pro zvětšující se úlohy zhruba konstantní. Pro SC-GPU1 se doba výpočtu zvětšovala.



Obrázek 7.23: Výstup nvprof pro výpočet 256x256x256 matice na dvou GPU zobrazený v Nvidia Visual Profileru.

Kapitola 8

Závěr

V rámci této práce byly implementovány a vyzkoušeny různé implementace pro akcelerování 3D FFT a tím celé simulace k-Wave[6], které zkoušeli využití unifikované paměti[3][5] a cuFFT[1] a knihovny cuFFTXt[1].

Testování unifikované paměti[3][5] na více GPU dopadlo hůře oproti implementacím které využívají jedno GPU. Pro úlohy na jednom GPU unifikovaná paměť[3][5] fungovala velice dobře a to i bez jakéhokoliv "prefetchingu" paměti.

Pro unifikovanou paměť[3][5] a cuFFT[1] nedošlo k žádnému zrychlení díky pomalé komunikaci mezi GPU přes unifikovanou paměť[3][5]. Je možné, že pro rychlejší výpočet by bylo potřeba celou implementaci knihovny změnit, aby co nejvíce výpočtů na GPU využívalo více GPU, aby data nemusela migrovat mezi GPU.

Pro částečnou implementaci pomocí knihovny cuFFTXt[1] došlo k malému zrychlení u dostatečně velké úlohy. Výsledky měření z této implementace předpokládají, že u úplné implementace by šlo implementovat ostatní části knihovny pracující s daty rozdělenými mezi více GPU bez penalizace na výkonu.

U strojů SC-GPU1 a Barbora šlo vidět rozdíl ve výkonu výpočtů na více GPU. Hlavně u cuFFTXt[1] byl typ propojení GPU rozhodující pro dobu výpočtu a zrychlení získaného výpočtem na více GPU.

Do budoucna by bylo vhodné ověřit zdali by úplná implementace cufftXt[1] a ostatních části knihovny k-Wave[6] potvrdila či ještě nezlepšila výsledky zjištěné v této práci. Také by bylo vhodné detailněji prostudovat mechanismus přesouvání stránek u unifikované paměti[3][5] a pokusit se efektivněji využívat "prefetchingu".

Literatura

[1] *Dokumentace k cuFFT a cuFFTXt*. [Online; navštíveno 23.05.2021]. Dostupné z: https://docs.nvidia.com/cuda/cufft/index.html.

[2] *NVLink*. Wikipedia: the free encyclopedia. [Online; navštíveno 23.05.2021]. Dostupné z: https://en.wikipedia.org/wiki/NVLink.

[3] Harris, M. *Unified Memory for CUDA Beginners*. [Online; navštíveno 23.05.2021]. Dostupné z: https://developer.nvidia.com/blog/unified-memory-cuda-beginners/.

[4] Nandapalan, N., Jaros, J., Rendell, A. a Treeby, B. Implementation of 3D FFTs Across Multiple GPUs in Shared Memory Environments. In:. Prosinec 2012, s. 167–172. DOI: 10.1109/PDCAT.2012.79. ISBN 978-0-7695-4879-1. Dostupné z: https://www.researchgate.net/publication/261338330_Implementation_of_3D_FFTs_Across_Multiple_GPUs_in_Shared_Memory_Environments.

[5] Sakharnykh, N. *Maximizing Unified Memory Performance in CUDA*. [Online; navštíveno 23.05.2021]. Dostupné z: https://developer.nvidia.com/blog/maximizing-unified-memory-performance-cuda/.

[6] Treeby, B. a Cox, B. K-Wave: MATLAB toolbox for the simulation and reconstruction of photoacoustic wave fields. *Journal of biomedical optics*. Březen 2010, sv. 15, s. 021314. DOI: 10.1117/1.3360308. Dostupné z: https://www.researchgate.net/publication/44589023_k-Wave_MATLAB_toolbox_for_the_simulation_and_reconstruction_of_photoacoustic_wave_fields.

Příloha A

Obsah přiloženého paměťového média

Na přiloženém DVD jsou uloženy veškeré zdrojové soubory, výsledky měření a manuál pro přeložení ukázkové aplikace. Zde je uveden přesný popis adresářové struktury, která je uložena na DVD disku:

- **bench/** – Program pro měření rychlosti unifikované paměti popsané v 3. kapitole.
- **k-Wave_CUDA_orig/** – Původní implementace CUDA k-Wave
- **k-Wave_CUDA_uni/** – Implementace k-Wave používající unifikovanou paměť a jedno GPU
- **k-Wave_CUDA_uni_mGPU/** – Implementace k-Wave používající unifikovanou paměť a implementaci 3D FFT pro více GPU
- **k-Wave_CUDA_uni_prefetch_mGPU/** – Implementace k-Wave používající unifikovanou paměť, implementaci 3D FFT pro více GPU a "prefetching"
- **k-Wave_CUDA_cuFFTXt/** – Implementace k-Wave používající cuFFTXt
- **docs/thesis** – Zdrojové LATEX soubory pro sestavení PDF souboru této práce, včetně veškerých zdrojů použitých v této práci
- **docs/measurements** – výsledky měření použitých v této práci
- **inputs/** – Vstupní HDF5 soubory pro simulace
- **makefiles/** – Makefile soubory pro sestavení na SC-GPU1 a Barboře

Příloha B

Manuál

B.1 Přeložení programu

- Pro překlad na SC-GPU1 a Barboře jsou nachystány 2 makefile soubory ve složce makefiles.
- Na strojích SC-GPU1 a Barboře je potřeba načíst moduly CUDA a HDF5 pomocí příkazu `ml CUDA HDF5`.
- Detailnější popis se nachází v souboru README.txt v kořenovém adresáři.

B.2 Nastavení počtu GPU

Pro nastavení počtu GPU je třeba upravit zdrojové soubory a program znova přeložit. Detailnější popis se nachází v souboru README.txt v kořenovém adresáři.

B.3 Spuštění programu

Spuštění programu je popsáno v kapitole 5.2 a v souboru README.txt.