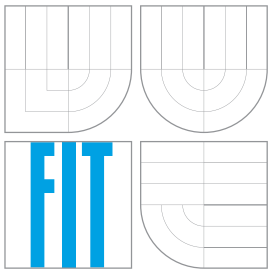
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

PARALELIZACE ULTRAZVUKOVÝCH SIMULACÍ NA SVAZKU GRAFICKÝCH KARET

PARALLELISATION OF ULTRASOUND SIMULATIONS ON MULTI-GPU CLUSTERS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. ALEŠ DUJÍČEK

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. JIŘÍ JAROŠ, Ph.D.

BRNO 2015

Abstrakt

Tato práce se zabývá rozšířením projektu k-Wave, který řeší simulace šíření ultrazvukových vln v heterogenním prostředí. Výpočet těchto simulací je založen na řešení soustavy parciálních diferenciálních rovnic pseudospektrální metodou.
Cílem této práce je využití lokální dekompozice pseudospektrální metody a výpočetního výkonu grafických karet ke zrychlení výpočtu těchto simulací. Dekompozicí výpočtu chceme dosáhnout nejen vyšší rychlosti, ale také možnosti provádět výpočet simulace ve větším prostoru, tedy s většími datovými mřížkami. Cílem je tedy dosáhnout nejen zrychlení, ale také dobré škálovatelnosti.

Abstract

This work is part of the k-Wave project, which is a toolbox designed for time ultrasound simulations in complex and heterogeneous media. The simulation functions are based on the k-space pseudospectral method. The goal of this work is to compute these simulations on graphics cards using local domain decompostion. Thanks to decomposition we could compute these simulations faster, and on larger data grids. The main goal of this work is to achieve efficiency and scalability.

Klíčová slova

ultrazvuková simulace, dekompozice, paralelizace, CUDA, OpenMPI, k-Wave, GPU

Keywords

ultrasound simulation, decomposition, parallelization, CUDA, OpenMPI, k-Wave, GPU

Citace

Aleš Dujíček: Paralelizace ultrazvukových simulací na svazku grafických karet, diplomová práce, Brno, FIT VUT v Brně, 2015

Paralelizace ultrazvukových simulací na svazku grafických karet

Prohlášení

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

.....
Aleš Dujíček
26. května 2015

Poděkování

Chtěl bych poděkovat vedoucímu práce Jiřímu Jarošovi za jeho velmi laskavý a motivující přístup a pomoc při řešení projektu.
Tato práce byla podpořena projektem Centrum excelence IT4Innovations (CZ.1.05/1.1.00/02.0070), který je financován Evropským fondem pro regionální rozvoj, ze státního rozpočtu České republiky v rámci operačního programu Výzkum a vývoj pro inovace a Ministerstvem školství, mládeže a tělovýchovy České republiky v rámci programu Projekty velkých infrastruktur pro VaVaI (LM2011033).

Obsah

Kapitola 1

Úvod

Simulace šíření ultrazvukových vln nachází uplatnění v mnoha oblastech, např. plánování zákroků ultrazvukem s vysokou intenzitou (HIFU) při léčbě nádorů. Těmito simulacemi se zabývá projekt k-Wave, který je představen ve 2. kapitole. Výpočet simulace je založen na řešení soustavy diferenciálních rovnic pseudospektrální metodou. [6]

Protože velikost oblasti tkáně, ve kterém se simulace počítá, je veliká a pro dostatečně přesný výpočet je potřeba velké množství bodů na jednotku délky, jsou kladeny velké nároky na paměť, ale je velká i výpočetní náročnost takové simulace.

K urychlení výpočtu využijeme výpočetního výkonu grafických karet, jejichž využití pro obecné výpočty je diskutováno v kapitole 3. Grafická karta má ale poměrně omezenou velikost paměti, už jen velikost samotné trojrozměrné matice čísel s jednoduchou přesností o velikosti $512 \times 512 \times 512$ dosahuje 512 MB, což spolu s dalšími daty potřebnými k výpočtu představuje více, než je kapacita paměti na grafické kartě.

Současná implementace, která využívá k výpočtu grafickou kartu, je asi 3krát rychlejší než implementace na procesoru [7]. Zapojení více grafických karet do výpočtu se ukázalo jako neefektivní z důvodu vysoké režie přenosu dat mezi kartami při výpočtu 3rozměrné rychlé Fourierovy transformace [11].

Pro praktické využití je ale potřeba provádět simulace s řádově většími počty bodů v mřížce a s rostoucí velikostí také roste výpočetní náročnost. Matici tedy rozdělíme na části a výpočet provedeme paralelně na svazku karet s využitím dekompozice pseudospektrální metody popsané v kapitole 5.

Paralelní výpočty budeme provádět na systémech s distribuovanou pamětí, k výměně dat mezi procesy v systému využijeme knihovnu pro zasílání zpráv Open MPI, která je blíže popsána v kapitole 4.

V kapitole 6 jsou podrobněji popsány jednotlivé části implementace lokální dekompozice pseudospektrální metody, která zahrnuje nutné výměny okrajů subdomén a samotný výpočet.

Experimentální měření vlastností implementace jsou pak shrnuty v kapitole 7, která je zaměřena především na využití teoreticky dosažitelné propustnosti komunikace a škálovatelnost výpočtu.

Kapitola 2

k-Wave

k-Wave je open source sada funkcí pro MATLAB vytvořená pro simulace šíření akustického signálu v čase v homogenním, nebo heterogenním prostředí v jedné, dvou nebo třech dimenzích. k-Wave také obsahuje implementaci v jazyce C++ k dosažení vyšší rychlosti simulace oproti MATLABu. [17]

Výpočet simulace je založen na řešení soustavy diferenciálních rovnic prvního řádu k-prostorovou pseudospektrální metodou. Tento nástroj také umožňuje výpočet zpětné rekonstrukce distribuce fotoakustického tlaku na základě zaznamenaných dat naměřených senzorem. [18]

Ultrazvukové simulace mají širokou škálu uplatnění v medicíně. Fotoakustická tomografie (PAT) je neinvazivní metoda vyšetření pacienta, která umožňuje vizualizovat struktury v tkáni absorbující světlo. Je založena na ozařování tkáně pulsy infračerveného laserového paprsku, který při absorpci v tkáni vytvoří ultrazvukový signál způsobený tepelnou roztažností látek. Měřením ultrazvukových vln, které se šíří zpět k povrchu tkáně, lze zrekonstruovat počáteční fotoakustický tlak. Tato metoda lze využít ke zjištění vlastností tkáně nebo detekovat patologické jevy v tkáni. [18]

HIFU (high-intensity focused ultrasound) je neinvazivní zákrok, kdy je ultrazvukový signál o vysoké intenzitě směřován do jednoho místa, kde způsobí zničení buněk. HIFU byl využit v klinických testech léčby nádorů různých orgánů. K naplánování takového zákroku slouží právě simulace šíření ultrazvukových vln, kterou k-Wave řeší. [6]

Průběh šíření akustických vln lze popsat soustavou diferenciálních rovnic:

$$\frac{\partial \mathbf{u}}{\partial t} = -\frac{1}{\rho_0}\nabla p \tag{2.1}$$

$$\frac{\partial \rho}{\partial t} = -\rho_0 \nabla \cdot \mathbf{u} \tag{2.2}$$

$$p = c^2 \rho \tag{2.3}$$

s počátečními podmínkami:

$$p_0 = \mathbf{\Gamma \mu_a \Phi},$$
$$\frac{\partial p_0}{\partial t} = 0$$

Kde $\mathbf{u}$ je akustická rychlost částice, ρ_0 je hustota prostředí, c je rychlost zvuku, p je akustický tlak, $p_0 = p(t = 0)$ je počáteční rozložení fotoakustického tlaku, Γ je Grüneinsenova konstanta (vztah mezi absorbovaným světlem a počátečním tlakem), μ_0 je koeficient

optické absorpce, Φ je světelný tok. k-Wave do těchto rovnic navíc zahrnuje i některé jevy, které způsobují nelinearitu v šíření. [18]

V současné době je implementace k-Wave v C++ s využitím OpenMP přibližně 8krát rychlejší než původní implementace v MATLABu a je možné s ní provádět simulace s doménou o velikosti 512^3 bodů, které trvají 2–3 hodiny. Verze k-Wave využívající MPI zvládne na systému se 2000 jádry vypočítat simulaci s doménou o velikosti 2048^3 za 50 hodin. Současná implementace v C++ využívající jednu grafickou kartu je přibližně 3krát rychlejší než C++ s OpenMP, je ovšem kvůli velikosti paměti grafické karty omezena velikost domény [7].

Naším cílem je využití dekompozice domény k rozdělení výpočtu mezi více grafických karet nejen k jeho urychlení, ale také umožnění provádět simulace i s většími rozměry domény. Takto by na superpočítačích jako Emerald s 384 grafickými kartami [15] nebo Titan s více než 18 tisíci grafickými kartami [9] mohlo být možné provádět simulace i ve velkých doménách s udržením doby výpočtu v řádu hodin.

2.1 Výpočet simulace šíření ultrazvukových vln

Výpočet simulace šíření ultrazvukových vln s využitím pseudospektrální metody je založen na výpočtu 3rozměrné dopředné a zpětné rychlé Fourierovy transformace a dalších maticových operacích jako násobení, dělení a sčítání matic po prvcích. Zde je zjednodušená časová smyčka simulace implementovaná v MATLABu: [7]

```
% smyčka kroků simulace v čase
for t_index = 2:Nt

    % výpočet 3D FFT akustického tlaku
1   p_k = fftn(p);

    % výpočet lokální rychlosti částic
2   ux_sgx = bsxfun(@times, pml_x_sgx,
        bsxfun(@times, pml_x_sgx, ux_sgx)
        - dt./rho0_sgx .* real(ifftn(
            bsxfun(@times, ddx_k_shift_pos, kappa .* p_k) ))
    );

    % přičtení vlivu ultrazvukového vysílače
3   if transducer_source >= t_index
      ux_sgx(us_index) = ux_sgx(us_index) +
        transducer_input_signal(delay_mask);
      delay_mask = delay_mask + 1;
    end

    % výpočet gradientu rychlosti částic
4   duxdx = real(ifftn(
        bsxfun(@times, ddx_k_shift_neg, kappa .* fftn(ux_sgx)) ));

    % výpočet akustické hustoty rhox
5   rhox = bsxfun(@times, pml_x,
        (rhox - dt.*rho0 .* duxdx) ./ (1 + 2*dt.*duxdx));
```

```
    % výpočet nového tlaku
6   p = c.^2.*( (rhox + rhoy + rhoz)
      + absorb_tau.*real(ifftn(
            absorb_nabla1 .* fftn(rho0.*(duxdx+duydy+duzdz)) ))
      - absorb_eta.*real(ifftn(
            absorb_nabla2 .* fftn(rhox + rhoy + rhoz) ))
      + BonA.*(rhox + rhoy + rhoz).^2 ./(2*rho0)
    );

    % extrakce a uložení dat z průběhu simulace
7   sensor_data(:, t_index) = p(sensor_mask_ind);
end
```

V bodě 1) je vypočtena 3rozměrná rychlá Fourierova transformace matice reprezentující akustický tlak. V bodě 2) jsou vypočítány nové hodnoty lokální rychlosti částic `ux_sgx` ve směru osy x, které popisují vibrace způsobené ultrazvukovými vlnami. Je zde využita funkce `bsxfun`, která provádí operaci s maticí a vektorem, v tomto případě násobení (`@times`), po prvcích tak, že vektor nejprve zopakuje v jednotlivých osách, aby byla vytvořena druhá 3rozměrná matice stejných rozměrů a mohla být provedena daná operace. V simulaci jsou pak analogicky vypočítány i rychlosti `uy_sgy` ve směru osy y a `uz_sgz` ve směru osy z. Dále je v bodě 3) započten vliv ultrazvukového vysílače na rychlost částic. Bodem 4) je vypočítán gradient `duxdx` lokálních rychlostí částic v prostoru ve směru osy x a opět analogicky i `duxdy` a `duxdz` pro osy y a z. Akustická hustota `rhox`, `rhoy` and `rhoz` je vypočítána v bodě 5) a na závěr je vypočítán nový tlak `p` v bodě 6 s využitím hodnot ze všech tří směrů os. Na závěr je v bodě 7) navzorkována matice tlaku a hodnoty jsou ukládány, aby mohl být sledován průběh simulace [7].

Právě v bodech 2), 4) a 6) je použita pseudospektrální metoda založená na výpočtu dopředné a zpětné FFT, kterou budeme řešit metodou lokální dekompozice. Ostatní maticové operace je možné také provádět jen v určité subdoméně, protože jsou prováděny po elementech.

Samotný výpočet pseudospektrální metody je v MATLABu implementován jako:

```
real(ifftn(bsxfun(@times, ddx_k_shift_pos, fftn(p))
```

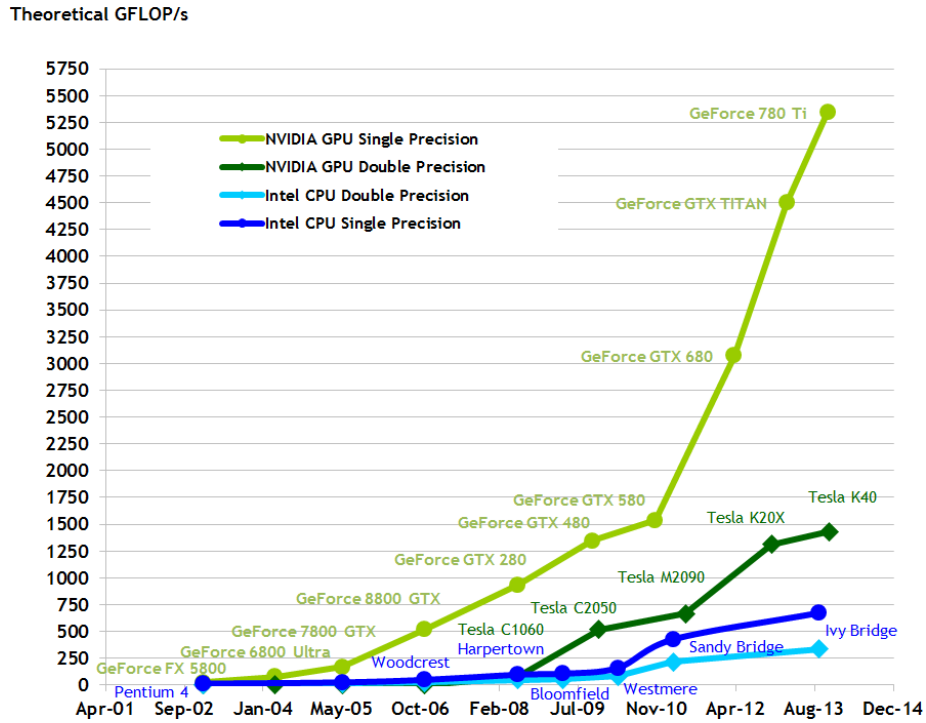
Nejdříve je vypočítáno spektrum pomocí 3rozměrné rychlé Fourierovy transformace, které je pak pomocí funkce `bsxfun` vynásobeno po elementech vektorem vlnových čísel `ddx_k_shift_pos`. Výsledkem je pak inverzní Fourierova transformace součinu spektra a vlnových čísel. Protože pracujeme pouze s reálnými signály, je na závěr imaginární část výsledku zpětné transformace zanedbána.

Kapitola 3

Obecné výpočty na grafických kartách

Grafické karty (GPU, graphics processing unit) byly původně vyvinuty ke generování grafického výstupu počítačů. S tím, jak časem narůstaly nároky na složitost vykreslovaných scén kvůli stále vyššímu rozlišení vykreslovaných scén, vyššímu počtu zobrazovaných snímků za sekundu a zobrazování 3D scén, rostl také rapidně výpočetní výkon těchto karet.

Tak časem vznikl velký rozdíl ve výpočetním výkonu (počet zpracovaných operací za jednotku času) běžných procesorů a grafických karet, což je znázorněno v grafu obrázku 3.1. Tak vznikla tendence tento výpočetní výkon využít pro obecné výpočty, označováno jako GPGPU (general-purpose computing on graphics processing units). [14]

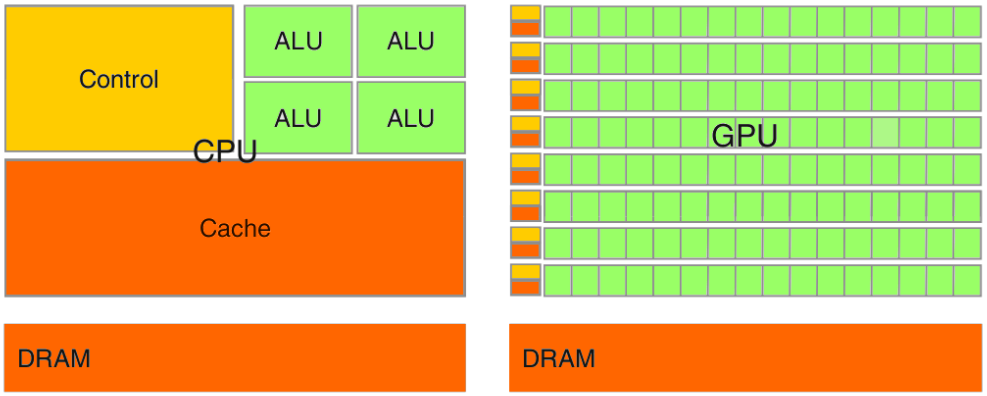


Obrázek 3.1: Porovnání výpočetního výkonu CPU a GPU [8]

Takto velký rozdíl ve výkonu CPU a GPU je dán jejich úplně rozdílnou architekturou, které jsou zobrazeny na obrázku 3.2. Procesor (CPU, central processing unit) je optimal-

izován k provádění sekvenčního kódu, k tomu je využito složité kontrolní logiky, která umožňuje vykonávat instrukce jednoho vlákna paralelně, nebo mimo pořadí. Další důležitou technikou zvýšený výkonu CPU je využití vyrovnávacích pamětí (cache). Všechny tyto obvody zabírají oproti aritmeticko-logickým jednotkám (ALU) velkou plochu čipu CPU.

Zatímco dnešní GPU je tvořeno velkým množstvím paralelních výpočetních jednotek, které provádějí kód mnoha vláken a překrývají tak velké latence přístupu do paměti. GPU tedy obsahují jen malé vyrovnávací paměti a jednoduchou řídící logiku a většina plochy čipu je věnována ALU. [8]



Obrázek 3.2: Rozdílná architektura CPU a GPU [8]

3.1 CUDA

CUDA (compute unified device architecture) je výpočetní platforma vyvinutá společností NVIDIA, která umožňuje využití výpočetních prostředků jejich grafických karet v programech napsaných v jazycích C, C++ a dalších. [13]

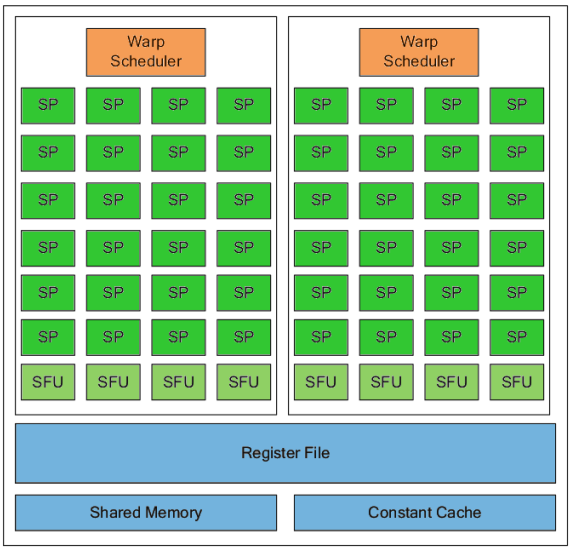
3.1.1 Architektura karet NVIDIA

Architektura karet NVIDIA se sice neustále mění, jak prochází vývojem, v principu ale zůstává velmi podobná. Jednotlivá jádra karty (tzv. streaming processor, resp. cuda core) jsou organizována spolu s dalšími jednotkami do větších celků zvaných streaming multiprocessor (SM), jak je zobrazeno na schématu mikroarchitektury Fermi na obrázku 3.3.

SM vedle samotných jader, které obsahují aritmetické jednotky pro čísla s plovoucí řádovou čárkou i celá čísla, obsahuje ještě další části: plánovač, který řídí spouštění vláken v SM jednotce, super function units (SFU), load/store jednotky (LS), registry, cache, paměť konstant a další. [19]

Protože jádra implementují jen základní aritmetické operace jako násobení a sčítání celých čísel a čísel s plovoucí řádovou čárkou a bitové operace nad celými čísly, slouží SFU k výpočtům dalších operací jako je dělení, goniometrické funkce a další. Load/store jednotky provádějí paměťové operace.

Počet výpočetních jader v jednom SM procesoru převyšuje počet ostatních zdrojů jako jsou load/store jednotky a jednotky SFU, jejich nadměrné využívání pak má negativní vliv na rychlost výpočtu, protože je nemohou využít všechna vlákna najednou. Paralelismus



Obrázek 3.3: Schéma architektury Fermi [19]

GPU je totiž realizován tak, že každé jádro SM jednotky vykonává kód jednoho vlákna, ale na všech jádrech se současně provádí stejná instrukce nad různými daty.

Úlohou plánovače je pak prokládání vykonávání různých skupin vláken, tzv. warpů, v SM jednotce, pro vykrývání velké latence přístupu do paměti výpočtem.

3.1.2 Paměťový model GPU

K dosažení co nejvyššího výkonu CUDA využívá různé typy pamětí, které jsou vhodné k různému využití. Paměť připojená k CPU se označuje jako host memory, zatímco paměť na GPU se označuje jako device memory. Vlákna na GPU mohou přistupovat pouze do paměti GPU, ale CUDA poskytuje funkce pro kopírování dat mezi těmito pamětmi a také umožňuje namapovat paměť CPU do adresového prostoru GPU.

Paměť na GPU tedy můžeme rozdělit do několika typů:

Globální paměť je staticky nebo dynamicky alokovatelná paměť, přístup k ní provádí load/store jednotky. Její součástí je také *lokální paměť*, do které se ukládají lokální proměnné vláken, které jsou typu pole.

Paměť konstant slouží pouze pro čtení, přístup do této paměti je prováděn speciální jednotkou, která umožňuje i čtení více vlákny současně.

Paměť textur, která slouží také jen pro čtení a podobně jako paměť konstant i paměť textur je obsluhována oddělenou jednotkou optimalizovanou pro čtení 1D, 2D a 3D struktur.

Sdílená paměť, která není součástí hlavní paměti, ale nachází se v SM procesoru. Tato paměť je sdílena mezi vlákny v bloku a může být použita k rychlé výměně dat mezi vlákny v bloku. Přístup do sdílené paměti je rychlejší než přístup do globální paměti a je důležitým prostředkem dosažení velké rychlosti výpočtu, zejména sdílením dat načtených z pomalejší globální paměti více vlákny a jejich opakovaným použitím.

Registry, které se také nacházejí přímo v SM jádře, jsou nejrychlejší dostupnou pamětí pro vlákna, slouží k ukládání lokálních proměnných.

Množství využitých registrů vlákny a velikost sdílené paměti v bloku limituje počet bloků současně přidělených jedné SM jednotce. Pokud má SM jednotka příliš málo bloků nemusí se jí podařit efektivně využít svých výpočetních prostředků, což se může negativně

projevit na výkonnosti aplikace. Proto je vhodné velikosti bloků a množství použité sdílené paměti vhodně volit vzhledem ke konkrétním hardwarovým parametrům dané grafické karty.

3.1.3 Spouštění kódu na GPU

Funkce spouštěné na grafické kartě se nazývají kernel. Ve zdrojovém kódu se tyto funkce určené k běhu na GPU v hlavičce jejich definice označují klíčový slovem `__global__` , nebo `__device__`. Vykonání kernelu na GPU je vhledem k CPU asynchronní, řízení je tedy hned po spuštění vráceno CPU a vykonání kernelu může probíhat paralelně s dalšími operacemi na CPU.

Ke spuštění kernelu pak slouží speciální syntaktická konstrukce,

```
kernel<<<blocks, threads>>>(...);
```

kde `kernel` je identifikátor funkce, `blocks` a `threads` jsou parametry spuštění, v závorce jsou volitelně parametry funkce.

CUDA seskupuje vlákna do bloků a bloky do mřížky (gridu), bloky i grid mohou být vícerozměrné. Každé vlákno pak má přístup k proměnným `threadIdx`, `blockIdx`, `blockDim` a `gridDim`, které určují číslo vlákna v bloku, číslo bloku v gridu, počet bloků v gridu a počet vláken v gridu v jednotlivých dimenzích. Tyto proměnné umožňují vláknům indexovat data v maticích, se kterými provádějí výpočet. Právě k definici rozměrů gridu a počtu vláken v bloku slouží zmíněné parametry `blocks` a `threads`. Při spouštění kernelu dále mohou být uvedeny další parametry, velikost sdílené paměti v bloku, pokud není známá už v době překladu, a stream, ve kterém má být kernel vykonán.

Takováto organizace vláken do bloků a bloků do gridu například umožňuje vytvořit mřížku vláken v prostoru, která odpovídá uspořádání dat v paměti, a tím zjednodušit adresování dat v maticích.

Bloky vláken jsou dále důležité z toho důvodu, že vlákna jsou na SM jednotky GPU přidělována právě po blocích. Na jednu SM jednotku může být současně přiřazeno více bloků. Pro každý přidělený blok jsou v SM jednotce alokovány zdroje, což limituje počet současně prováděných bloků na jedné SM jednotce. Vlákna spolu navíc mohou v rámci bloku komunikovat pomocí sdílené paměti a mohou být synchronizována bariérou, vzájemná synchronizace bloků není možná.

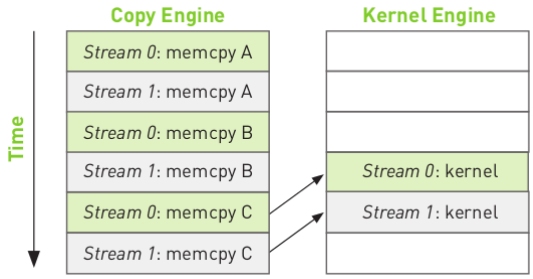
Bloky jsou tedy přiděleny SM jednotce, která vykonává paralelně pouze část vláken z bloku, tzv. warp. Velikost warpu odpovídá počtu jader v SM procesoru, v současnosti je velikost warpu 32 vláken. Jednotou provádění instrukcí je tedy warp. SM jednotky přidělují vláknům prostředky po warpech a v jeden okamžik je v celém warpu vykonávána stejná instrukce.

V případě zpracování podmíněných příkazů je velmi žádoucí, aby v rámci warpu byly vyhodnoceny stejně, aby nedocházelo k tomu, že různá vlákna ve warpu mají provádět jiné instrukce. Tento nežádoucí jev se označuje jako divergence vláken, která může mít negativní dopad na výkonnost, protože pokud k ní dojde, tak musí být provedeny obě větve podmíněného příkazu.

3.1.4 CUDA streams

Stream v CUDA je prostředek, který umožňuje současný běh některých úloh na grafické kartě. Implicitně jsou všechny operace spouštěny v takzvaném *null stream* a jsou spouštěny

všechny sekvenčně. CUDA stream je vlastně fronta úloh, které jsou spuštěny sekvenčně, ovšem těchto front je možné vytvořit více, a pak je možné dosáhnout současného běhu úloh z různých front, které jsou prováděny různými hardwarovými jednotkami na grafické kartě [19].



Obrázek 3.4: Běh více operací v různých CUDA streamech[14]

Na obrázku 3.4 je příklad využití dvou CUDA streamů k překrytí spuštění dvou kernelů a paměťových přenosů vstupních (A a B) a výstupních dat (C) těchto kernelů, kdy je samotný výpočet v čase plně překryt paměťovými přenosy. Podobného principu překrytí bude výhodné využít při výpočtu, kdy bude potřeba přenášet poměrné velké množství dat kvůli výměnám okrajů subdomén dekompozice.

Kapitola 4

Open MPI

Open MPI je knihovna implementující funkce standardu MPI [4]. Message Passing Interface (MPI) je standard popisující aplikační programové rozhraní (API) pro komunikaci procesů v distribuovaných paralelních systémech formou zasílání zpráv.

MPI je vytvořeno pro systémy s distribuovanou pamětí, kde každý uzel má vlastní lokální paměť a sdílení dat je možné pouze formou zasílání zpráv. Výhodou je, že není nutná synchronizace procesů, protože každý proces má vlastní paměťový prostor a nemůže tedy docházet k souběžnému přístupu ke sdíleným prostředkům. Nevýhodou tohoto přístupu pak může být velká režie komunikace mezi procesy.

MPI nachází uplatnění v oblasti vysoce náročných výpočtů (high-performace computing) při vytváření aplikací pro clustery, které mají právě architekturu založenou na distribuované paměti.

Cílem MPI je vytvoření standardu s ohledem na nezávislost na platformě a programovacím jazyce, efektivitu, vysoký výkon, jednoduchost použití, použití v heterogenních systémech. [10]

MPI popisuje celou řadu prostředků pro programování paralelních aplikací jako point-to-point komunikaci, kolektivní operace, datové typy, skupiny a topologie procesů, jednostrannou komunikaci atd.

4.1 MPI procesy a point to point komunikace

Spuštění paralelní úlohy, která využívá MPI se provádí nástrojem `mpiexec`, kterým se nastavují parametry běhu distribuované aplikace, tedy především kolik procesů se vytvoří, na jakých uzlech a podobně.

Každý MPI proces před použitím jakýchkoliv funkcí musí zavolat inicializační funkci `MPI_Init` a před ukončením procesu `MPI_Finalize`. Po spuštění aplikace se všechny procesy nachází v jednom komunikátoru `MPI_COMM_WORLD`, v rámci něhož mají unikátní identifikátor tzv. rank.

Základní možností komunikace procesů je point to point komunikace operacemi *send* a *receive*. Operace *send* má jako parametr ukazatel na data, datový typ, počet položek k odeslání a rank příjemce, *receive* analogicky rank odesílatele, ukazatel na bufffer pro uložení dat, jejich typ a maximální počet přijatých položek.

Operace *send* a *receive* mají podobně jako jiné operace blokující i neblokující varianty.

MPI definuje některé důležité vlastnosti komunikace. Zprávy se nepředbíhají, pořadí zaslaných zpráv zůstává zachováno. Pokud některý proces odešle stejnému příjemci více

zpráv, pak jsou doručeny ve stejném pořadí, jako byly odeslány. Dále je to garance, že pokud byly zahájeny dvě odpovídající operace *send* a *receive*, pak alespoň jedna z nich skončí. MPI negarantuje spravedlnost, tedy pokud je odesláno více zpráv stejnému příjemci, není garantováno, že zprávy z jednoho zdroje nebudou neustále předbíhány jinými.

4.2 Kolektivní operace procesů

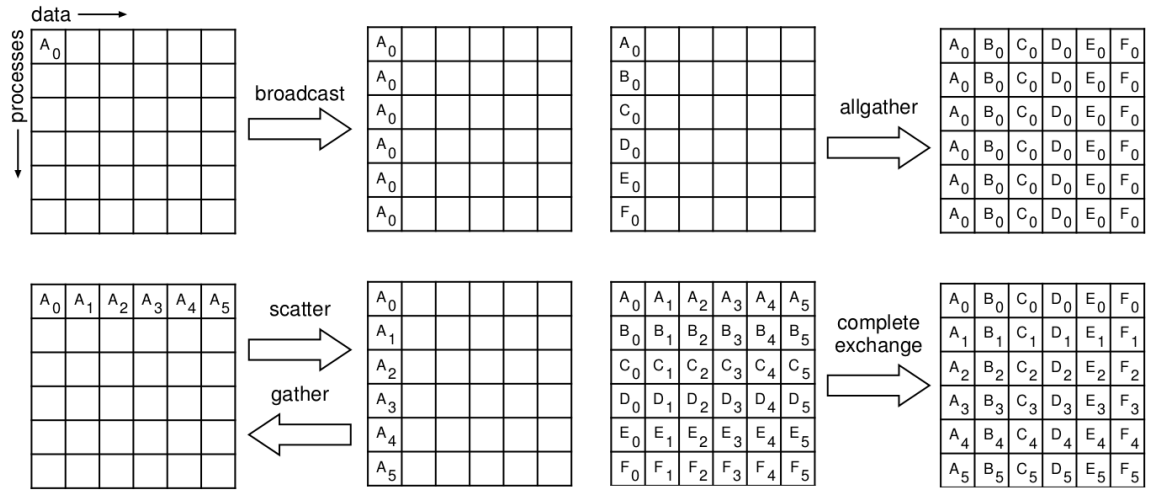
MPI definuje nejen point to point komunikace, ale zavádí také kolektivní operace, do kterých může být obecně zapojeno více procesů v rámci komunikátoru. Komunikátory zastřešují nějakou skupinu procesů, které spolu mohou komunikovat, a jejich topologii.

Mezi kolektivní operace patří například *scatter*, *gather*, *broadcast* a další. Operace *scatter* z jednoho uzlu distribuuje data do ostatních. To je výhodné například v případě, kdy jeden uzel má přístup k souboru s daty a potřebu je rozdistribuovat mezi ostatní uzly a není tak potřeba vytvářet velké množství jednotlivých přenosů dat ke všem uzlům. Tuto operaci tak využijeme při distribuci dat jednotlivým procesům při dekompozici vstupní matice dat.

Operace *gather* naopak sesbírá data ze všech uzlů do jednoho, obdobně lze využít pro uložení výsledků na konci výpočtu, což využijeme také. Obě dvě operace mají také varianty, které umožňují definovat různý počet datových položek odesílaných, resp. přijímaných dat, do každého procesu. Další variantou je operace `allgather`, která sesbíraná data navíc doručí všem procesům.

Operace *alltoall* provádí úplnou výměnu dat mezi procesy.

Funkce těchto operací je znázorněna na obrázku 4.1).



Obrázek 4.1: Kolektivní komunikace MPI [10]

Podobně jako u point to point operací i zde jsou zavedeny také neblokující varianty těchto operací, které umožňují překrýt výpočet a komunikaci k dosažení vyššího výkonu.

Další kolektivní operací je synchronizační bariéra *barrier*, která je blokující do okamžiku, kdy všechny procesy v komunikátoru tuto operaci nezavolají. K rozeslání stejných dat z jednoho procesu všem ostatním slouží operace *broadcast*.

MPI také definuje kolektivní operace redukce *reduce* a *scan*, které nad daty provádí nějakou operaci např. součet, součin nebo maximum. MPI obsahuje řadu předdefinovaných operací a umožňuje definovat i vlastní, taková operace musí být asociativní, ale už ne nutně komutativní.

4.2.1 Sousedské komunikace

MPI umožňuje definovat grafovou a kartézskou topologii procesů, která určuje jejich okolí, v rámci kterého pak procesy mohou provádět sousedské kolektivní operace. Definované sousedské operace jsou obdobné těm popsaným výše.

Tento koncept zkusíme využít při výměně dat na okrajích matic, které si během výpočtu budou uzly vyměňovat.

4.3 Datové typy

Pro zjednodušení práce s daty a k zefektivnění jejich přenosu zavádí MPI možnost odvozovat nové datové typy. Takto je možné posílat zprávy obsahující heterogenní nebo nesouvislá data, ty by sice bylo možné nejdříve překopírovat do spojitého bufferu a odeslat běžným způsobem, ale je vyžaduje to zbytečnou režii.

Nejjednodušším datovým typem je typ *contiguous*, který vytváří spojité pole hodnot odvozovaného typu. Typ *vector* vytváří nespojité pole stejně dlouhých bloků odvozeného typu, které mezi sebou mají stejné rozestupy. Tento typ lze využít k definování typu pro sloupce dvourozměrného pole, které má v paměti uloženy spojitě řádky.

Typ *indexed* je obecnější varianta typu *vector*, umožňuje navíc definovat různé délky jednotlivých bloků i různé rozestupy mezi bloky.

Nejobecnější je typ *struct*, který umožňuje, aby byl každý blok tvořen položkami různých typů. Vytváří tak heterogenní strukturu.

Zajímavou možností je vytvoření typu *subarray*, který umožňuje vytvořit n-rozměrné podpole nějakého n-rozměrného pole, které v něm může být libovolně umístěno. Tento typ využijeme při výměně okrajů matic, kdy budeme při vícerozměrné dekompozici potřebovat definovat různé oblasti dat pro výměnu.

Kapitola 5

Dekompozice pseudospektrální metody

Simulace šíření ultrazvukových vln v heterogenním prostředí je založena na výpočtu disktretizované soustavy parciálních diferenciálních rovnic, popsaných v kapitole 2, pseudospektrální metodou. Tato metoda je založena na výpočtu 3rozměrné rychlé Fourierovy transformace. Místo rozdělení této transformace do distribuovaného výpočtu 1rozmerných transformací ve směru jednotlivých os, což přináší velkou režii komunikace při transformacích matice, rozdělíme doménu na menší části a výpočet rychlé Fourierovy transformace tak bude prováděn jen s lokálními daty. Pseudospektrální metoda a její dekompozice je blíže popsána v následujících podkapitolách.

5.1 Pseudospektrální metoda

Parciálními diferenciálními rovnicemi se popisuje mnoho fyzikálních jevů a jejich modelování je založeno na jejich numerickém výpočtu, protože přesné analytické řešení by bylo příliš složité.

Metoda konečných diferencí (FD) a metoda konečných elementů (FE) jsou běžnými numerickými metodami řešení parciálních diferenciálních rovnic. Metoda konečných diferencí aproximuje derivace funkce pomocí okolí bodu (např. $\frac{du}{dx} = \frac{u(x+h)-u(x-h)}{2h}$), kde h je malý krok v mřížce. Dochází tedy k interpolaci polynomem, čím více bodů z okolí použijeme, tím vyššího řádu polynomu dosáhneme a také vyšší přesnosti.

Z toho také vyplývají omezení FD metody. Pro dostatečnou přesnost výpočtu je potřeba malý krok mezi body v mřížce, což způsobuje velké nároky na paměť a také vysokou výpočetní náročnost.

Tento problém řeší pseudospektrální metoda (PS), která nezakládá výpočet jen na nějakém malém okolí bodu, ale přistupuje k funkci globálně. Funkci, kterou chceme derivovat, PS metoda aproximuje sumou nějakých spojitých bázových funkcí, v našem případě komplexními exponenciálami. Takto se dosáhne dobré přesnosti výpočtu i nižším počtu bodů v mřížce než při použití metod konečných prvků nebo konečných diferencí [3]

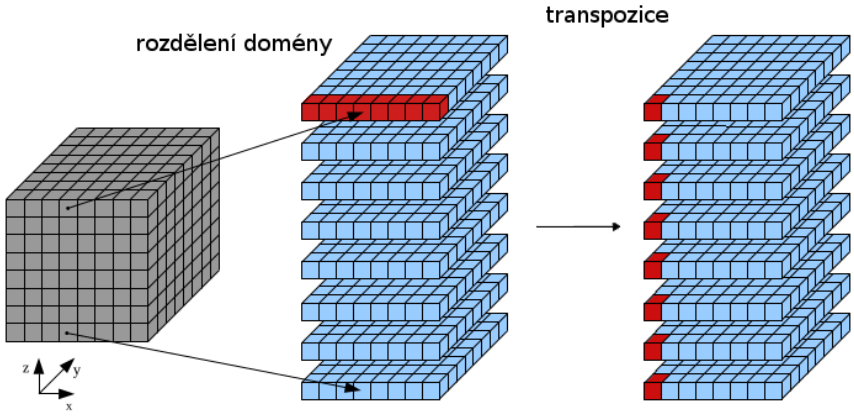
Pseudospektrální metoda má další výhodu, lze ji implementovat s využitím rychlé Fourierovy transformace (FFT), která má mnoho optimalizovaných implementací, my využijeme knihovnu cuFFT – implementace FFT pro CUDA.

Výpočet gradientu v prostoru pseudospektrální metodou lze shrnout do třech kroků: [3]

- výpočet FFT vstupních dat

- vynásobení výsledku prvního kroku maticí vlnových čísel
- výpočet inverzní FFT

Výpočet gradientu pseudospektrální metodou je tedy založen na Fourierově transformaci, která se efektivně vypočítá pomocí algoritmu FFT. Výpočet vícerozměrné FFT je proveden jako řada 1D FFT podél jednotlivých os, což umožňuje doménu rozložit na části a počítat jednotlivá 1D FFT paralelně. Na obrázku 5.1 je znázorněno rozdělení 3D matice podél osy z mezi jednotlivé procesy distribuovaného výpočtu. Každý proces tak může paralelně vypočítat FFT podél os x a y. Pro výpočet 1D FFT podél osy z je ale nutné použít data distribuovaná přes všechny procesy, což vede k all-to-all komunikaci při transformaci matice. All-to-all komunikace představuje velkou režii a zhoršuje škálovatelnost, protože počet zpráv při all-to-all komunikaci roste kvadraticky s počtem uzlů. Dekompozice tímto způsobem je také limitována rozměry matice, kdy počet procesů musí být menší nebo roven rozměru matice v ose z. [6]



Obrázek 5.1: Dekompozice 3D matice a transpozice [6]

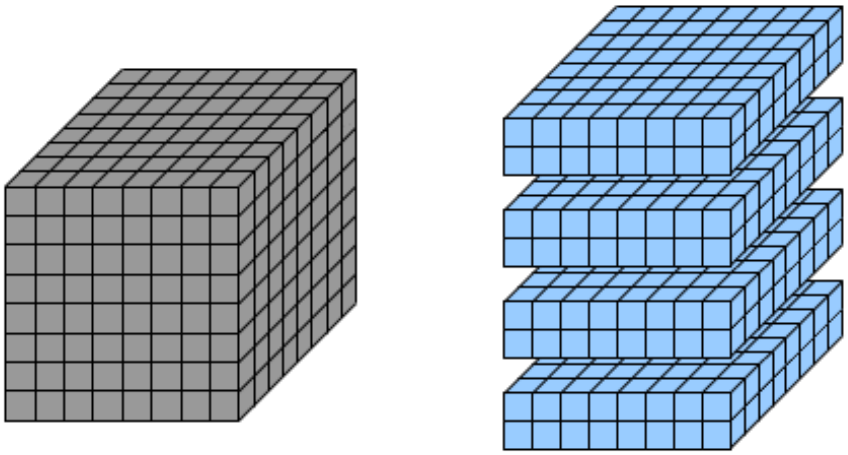
Tento problém se pokusíme vyřešit výpočtem gradientu metodou lokální dekompozice pseudospektrální metody. Tímto přístupem omezíme režii komunikace při výpočtu pouze na výměnu dat na okrajích matic mezi sousedy.

5.2 Lokální dekompozice pseudospektrální metody

Pro dosažení vysoké rychlosti výpočtu a velké škálovatelnosti budeme chtít rozdělit výpočet mezi procesy tak, aby každý počítal pouze s částí vstupních dat. Na začátku se omezíme jen na 1D dekompozici, kdy data rozdělíme podél jedné z os do stejně velkých bloků, jejichž počet odpovídá počtu procesů, jak je ukázáno na obrázku 5.2.

Abychom takto mohli výpočet rozdělit, bude nutné ukázat, že řešení pseudospektrální metody omezená na podinterval definičního oboru funkce odpovídá v tomto intervalu řešení nad celým definičním oborem.

Toto je ukázáno v [5], kde řešení funkce f pseudospektrální metodou omezí pouze na podinterval (a, b). Pro výpočet je nutné dosáhnout hladkého průběhu funkce na okrajích



Obrázek 5.2: 1D dekompozice domény

intervalu, rozšíří proto interval (a, b) na interval $(a_1, b_1), a_1 < a < b < b_1$. A pro vyhlazení definují ještě takzvanou bell funkci B,

$$B^2(x) + B^2(2\bar{a} - x) = 1, x \in (a_1, a) \tag{5.1}$$

$$B(x) = 1, x \in < a, b > \tag{5.2}$$

$$B^2(x) + B^2(2\bar{b} - x) = 1, x \in (b, b_1) \tag{5.3}$$

$$B(x) = 0, x < a_1, x > b_1 \tag{5.4}$$

kde $\bar{a} = (a + a_1)/2, \bar{b} = (b + b_1)/2$. Vyhlazením pak dostaneme funkci $\tilde{f}$,

$$\tilde{f} \equiv B \cdot f = B(x)f(x) - B(2\bar{a} - x)f(2\bar{a} - x) - B(2\bar{b} - x)f(2\bar{b} - x) \tag{5.5}$$

která je spojitá a hladká v bodech a a b. Aplikací pseudospektrální metody na funkci $\tilde{f}$ dostaneme v intervalu (a, b) řešení odpovídající řešení na funkci f v celém definičním oboru. [5]

Rozšíření intervalu (a, b) na (a_1, b_1) nám definuje nějaký překryv částí domény funkce po dekompozici, nicméně teď víme, že můžeme pseudospektrální metodu řešit po částech nezávisle až na tento překryv.

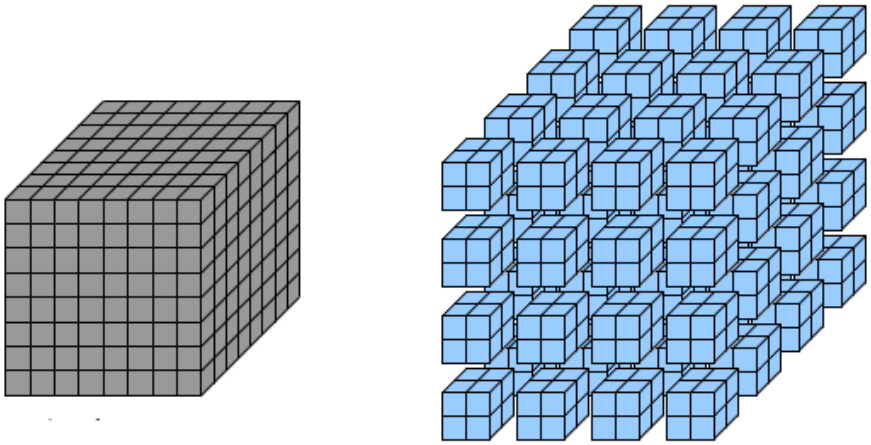
5.3 Další možnosti dekompozice

Lokální dekompozicí výpočtu dosáhneme oproti globálnímu výpočtu snížení režie spojené s all-to-all komunikací při paralelním zpracování jednotlivých 1D FFT, jak je popsáno v předchozí podkapitole. Režie komunikace bude snížena jen na výměnu okrajů jednotlivých subdomén.

Pokud bychom měli výpočet nad datovou krychlí o straně délky N rozdělenou 1D dekompozicí mezi P procesů, pak při překrytí o bude v každém kroku výpočtu docházet k výměně celkem $2P$ bloků dat o velikosti oN^2. Bude tedy docházet k malému počtu zaslaných zpráv, které budou poměrně veliké. Tento přístup nám ovšem limituje možný počet paralelních procesů P, který musí být výrazně menší než N.

Pokud přistoupíme ke 2D dekompozici, tedy že datovou kostku rozdělíme ve dvou osách na m a n částí mezi celkem $P = m \cdot n$ procesů. Pro jednoduchost budeme uvažovat $m = n$. Pak bude velikost bloků dat na okrajích $oN^2 \cdot \frac{\sqrt{P}}{P}$, ovšem výměna tentokrát bude probíhat ve 4 směrech, takže se zvýší počet výměn těchto bloků na $4P$. Navíc ještě přibude $4P$ výměn datových bloků o velikosti o^2N, které jsou nutné pro doplnění dat v rozích okolí.

Obdobně můžeme zajít až ke 3D dekompozici mezi $P = k \cdot m \cdot n$ procesů, pro jednoduchost budeme opět uvažovat $k = m = n$. 3D dekompozicí rozdělíme doménu na celkem P podkostek, jak je znázorněno na obrázku 5.3. V takovém případě stoupne počet zaslaných zpráv na $6P$, které ale budou mít velikost $oN^2 \cdot \frac{\sqrt[3]{P}}{P}$. Navíc přibude $8P$ zpráv o velikosti o^3 nutných opět k výměně dat v rozích a 12 zpráv o velikosti $o^2N\frac{\sqrt[3]{P^2}}{P}$ pro data na hranách datové kostky.



Obrázek 5.3: 3D dekompozice

Využitím 2D a 3D dekompozice dosáhneme možnosti využití více paralelních jednotek a zmenší se velikost dat překryvů, protože krychle má nejmenší povrch při daném objemu. Na druhou stranu se tím ovšem zvýší počet zasílaných zpráv v každém kroku výpočtu. Bude tedy otázkou, jak to ovlivní škálovatelnost a rychlost výpočtu.

Kapitola 6

Návrh a implementace

Implementace simulace šíření ultrazvukových vln s použitím metody lokální dekompozice pseudospektrální metody vyžaduje vedle samotného výpočtu, který již v k-Wave implementován je, vyřešit navíc výměny okrajů, jednotlivých subdomén dekompozice, což zahrnuje nejen jejich samotný přenos mezi uzly, ale také přenos z grafické karty do hlavní paměti a zpět.

Protože cluster Anselm, na kterém jsem mohl aplikaci testovat, obsahuje výpočetní uzly s jedním grafickým akcelerátorem, byla aplikace navržena tak, že celá doména se rozdělí na jednotlivé subdomény, kdy každý uzel bude provádět výpočty na akcelerátoru nad jednou touto subdoménou.

Výpočet šíření ultrazvukových vln se skládá z operace nad maticemi po elementech, které lze provádět na jednotlivých subdoménách nezávisle na sobě a hlavně z výpočtů pseudospektrální metodou, kde je již potřeba komunikovat s okolními uzly, tento výpočet se pak bude skládat z několika kroků, které jsou blíže popsány v následujících podkapitolách:

- přeskládání dat okrajů do lineární paměti
- kopie dat do paměti CPU
- použití MPI k výměně dat s okolními uzly
- kopie dat do paměti GPU
- přeskládáni dat okrajů z lineární paměti
- samotný výpočet

Nejvyšší flexibility je dosaženo použitím 3D dekompozice, kdy je možné volit, na kolik částí je doména rozdělena ve všech třech osách. Pro výkonnost je totiž nejvýhodnější, když jednotlivé subdomény tvoří krychle nebo se krychli alespoň blíží, protože v takovém případě se dosáhne nejlepšího poměru mezi velikostí samotných dat a velikostí okrajů. Z toho pak plyne, že je výpočet prováděn nad velkým množstvím dat a současně je nižší režie spojená se zvětšením subdomény o okraje od okolních subdomén. Např. při dekompozici domény o velikosti 512^3 bodů do 8 subdomén by při 1D rozdělení $8 \times 1 \times 1$ vznikly subdomény o velikosti $64 \times 512 \times 512$. Přidáním okrajů o šířce 16 bodů se subdoména zvětší na $96 \times 512 \times 512$. Okraje v tomto případě tvoří 50% z velikosti subdomény bez okrajů. Zato by při uplatnění 3D rozdělení $2 \times 2 \times 2$ byly rozměry subdomén 288^3, okraje jen 42% velikosti subdomény bez okrajů.

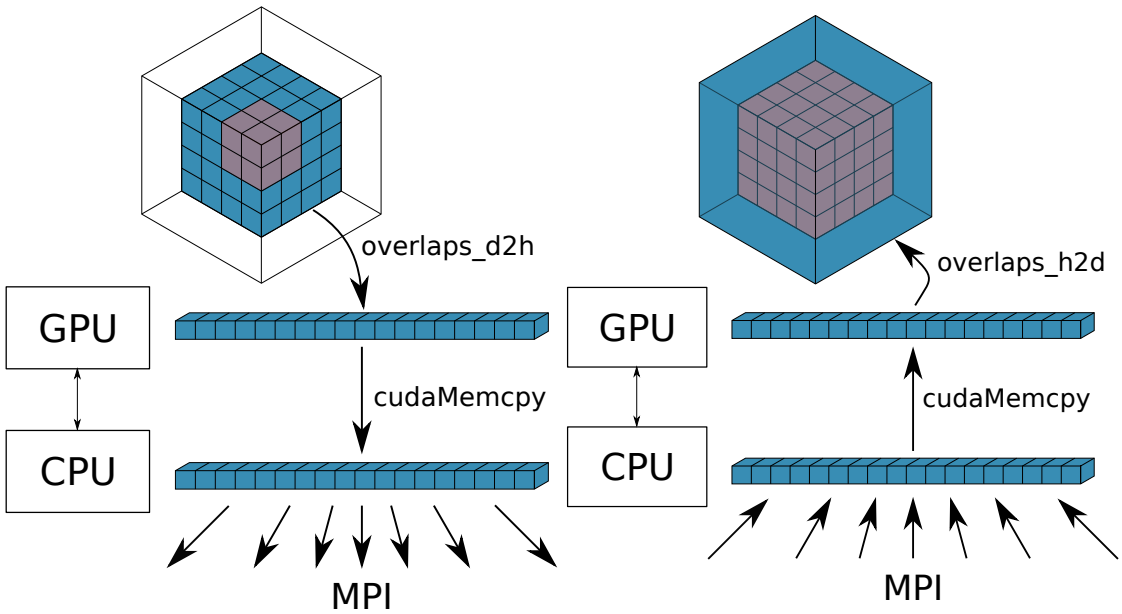
Stejně jako v k-Wave bude použit jako vstup i výstup implementace výpočtu gradientu pseudospektrální metodou souborový formát HDF5, který také umožní snadné ověření správnosti výpočtu oproti implementaci v MATLABu.

6.1 Přeskládání okrajů v paměti GPU

Pro přenos dat okrajů do hlavní paměti z grafické karty i opačným směrem je výhodnější, pokud jsou data v jednom velkém lineárním bloku. Pak není potřeba provádět neefektivní velké množství malých přenosů. Proto jsou před samotným přenosem data okrajů překopírována do jednoho lineárního pole. Navíc se jednotlivé bloky okrajů částečně překrývají, a to zejména v rozích, což je také vyřešeno zduplikováním těchto překrytí společně s přeskládáním, takže samotná data jsou pak již přímo připravena pro MPI komunikaci.

Pro toto přeskládání jsem zvažoval dvě varianty implementace. První možnost byla, že by kernel pokaždé přepočítával souřadnice datové kostky a lineárního pole s okraji, aby přesouval data na správná místa. Druhou variantou bylo tyto souřadnice pro každý bod předpočítat jen jednou, a pak je vždy jen použít. Nakonec jsem zvolil druhou z možností, přestože toto řešení zvýší počet přístupů do paměti, nicméně pak není potřeba provádět pořád dokola stejné aritmetické operace při přepočítávání souřadnic, což vede i k mnohem jednodušší implementaci kernelu, protože vzhledem k počtu různých bloků okrajů bylo nutné kód poměrně hodně větvit. Cenou za takové řešení je ovšem vyšší paměťová náročnost.

Celkové schéma provádění výměn okrajů je znázorněno na obrázku 6.1. Nejdříve kernel `copy_overlaps_d2h`, přerovná data okrajů z vnitřní části datové kostky, která odpovídá původní subdoméně bez přidaných okrajů, pro sousední uzly do lineárního pole, které je pomocí `cudaMemcpy` překopírováno do hlavní paměti, aby mohlo být následně pomocí MPI rozesláno sousedním uzlům. Z důvodu použití neblokující varianty `cudaMemcpyAsync`, bylo nutné před samotnou MPI komunikaci ještě doplnit synchronizaci, aby se začala odesílat data až v okamžiku, kdy jsou připravena.



(a) Kopie okrajů z GPU na CPU

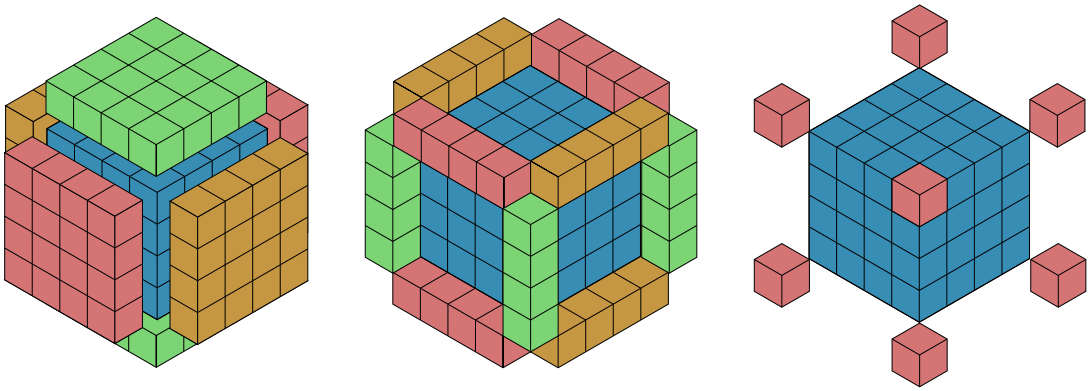
(b) Kopie okrajů z CPU na GPU

Obrázek 6.1: Výměny okrajů

Po dokončení asynchronních MPI přenosů byla opět nutná synchronizace, aby mohla být přijatá data od sousedních uzlů opět nakopírována do paměti grafické karty a za použití kernelu `copy_overlaps_h2d` přesunuta do datové kostky do oblasti, která odpovídá právě jejímu rozšíření o okraje. Žádná jiná synchronizace nebyla potřeba, operace, které provádí grafická karta jsou asynchronní jen z pohledu hostitelské aplikace, jinak jsou ale prováděny sekvenčně.

6.2 Výměny okrajů s MPI

Při uplatnění 3D dekompozice je okolí každé subdomény složeno nejen z šesti stěn přilehlých subdomén, to by v rozšíření subdoméně chyběla data v rozích a na hranách, k jejich úplnému doplnění jsou potřeba data i z dalších uzlů, se kterými sousedí přes hranu nebo roh, jak se zobrazeno na obrázku 6.2. Celkem je tedy potřeba získat data ze všech 26 uzlů v okolí.

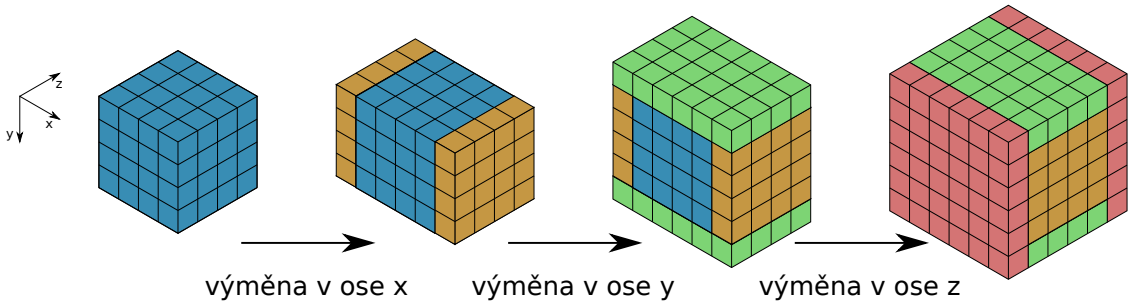


Obrázek 6.2: Rozšíření subdomény o překryv z okolních subdomén

Jednou z možností využití MPI je definování kartézské topologie, která ale vytvoří okolí složené jen právě z těch šesti uzlů, které přímo sousedí přes stěnu. Takto by bylo možné pomocí alltoall sousedské komunikace provést výměnu jen této části dat okrajů, nicméně by stále bylo potřeba doplnit ostatní chybějící data jiným způsobem. Toho by bylo možné dosáhnout rozdělením výměny na tři fáze, jak je znázorněno na obrázku 6.3. Kde by se v každé z těchto fází provedla výměna dat mezi sousedy jen v jedné ose. Takto by došlo k postupnému doplnění chybějících dat do zbylé části okrajů a všechna potřebná data by byla vyměněna jen v rámci této topologie.

Nicméně takové řešení by vyžadovalo nežádoucí synchronizaci mezi jednotlivými fázemi výměny okrajů. Navíc by také bylo zřejmě potřeba pokaždé provést nějaké přeskládání dat, aby mohla být provedena další fáze výměny, protože přijatá data z předchozí fáze by bylo potřeba dodatečně doplnit do připraveného bloku pro výměnu ve směru další osy.

Z těchto důvodů jsou kraje rozděleny do celkem 26 bloků, kde každý z jich je určen pro jiného souseda v okolí. Jde o celkem 6 bloků stěn, 8 rohů a 12 hran, jak je vidět na obrázku 6.2. K výměně bloků okrajů stěn jsem využil asynchronní variantu zmíněné alltoall sousedské operace v kartézské topologii (`MPI_Ineighbor_alltoallv`), pro výměnu ostatních bloků, už jsou použity jen asynchronní operace `MPI_Isend` a `MPI_Irecv`, protože by bylo potřeba vytvořit poměrně složitou obecnou grafovou topologii, aby mohly být také použity sousedské komunikace.



Obrázek 6.3: Výměny ve třech krocích

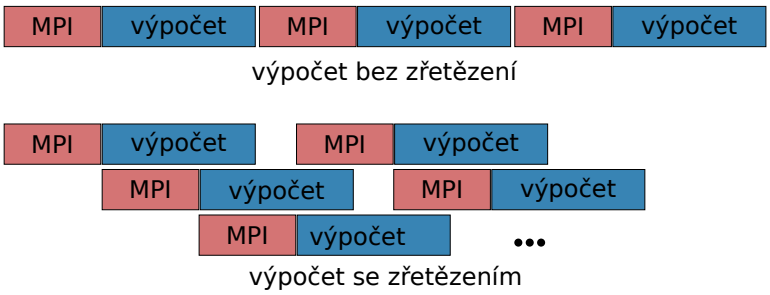
6.3 Implementace výpočtu

Po uskutečnění výměn okrajů se sousedy a jejich nahrání do paměti karty bude možné konečně provést výpočet gradientu pseudospektrální metodou, který je složen z výpočtu rychlé Fourierovy transformace a násobení matic po elementech. Výpočet je podrobněji popsán v kapitole 2.1. Při použití lokální dekompozice se tento výpočet změní na

```
real( ifftn( ddx_k_shift_pos .* fftn(bsxfun(@times, bell, p)) ) );
```

Kde je navíc před výpočtem spektra matice vynásobena bell funkcí, která kvůli přesnosti vyhladí data v okrajích.

Pseudospektrální metoda se v simulaci uplatní během jednoho kroku celkově sedmkrát. Zvlášť ve směrech os x, y a z pro matice akustických rychlostí ux, uy, uz a jejich gradientů $dudx$, $dudy$, $dudz$. Protože jsou tyto výpočty v jednotlivých osách na sobě nezávislé, pokusíme se překrýt komunikaci s výpočty, jak je znázorněno na obrázku 6.4. Tím by se měl snížit vliv režie spojené s výměnami okrajů a mělo by se dosáhnout vyšší rychlosti výpočtu. Nakonec je ještě jednou pseudospektrální metoda použita pro výpočet akustického tlaku v příštím kroku, ale ta je již datově závislá na datech vypočtených ve směrech všech tří os a pravděpodobně sníží efektivitu tohoto zřetězení.



Obrázek 6.4: Zřetězení výpočtu ve třech osách

K výpočtu rychlé Fourierovy transformace je využita knihovna cuFFT [12], která podporuje výpočet FFT nad komplexními daty (C2C – complex to complex) i reálnými daty (R2C, C2R – real to complex, complex to real). Protože jsou data v simulaci šíření ultrazvukových vln pouze reálná, je vhodné použít R2C a C2R transformace, protože se tak

ušetří značná část paměti, protože v tomto případě je spektrum symetrické a paměti tak stačí jen téměř polovina.

Samotné použití knihovny je takové, že se nejdříve vytvoří plán, v případě 3D FFT je to funkce `cufftPlan3d`, která má jako parametry rozměry datové matice a typ prováděné transformace, `CUFFT_R2C` pro dopřednou FFT a `CUFFT_C2R` pro zpětnou. Pro výpočet pseudospektrální metody jsou tedy potřeba oba tyto plány. Provedení samotného výpočtu se pak spustí funkcí `cufftExecR2C`, pro dopřednou transformaci, resp. `cufftExecC2R` pro zpětnou transformaci.

Implicitně provádí knihovna cuFFT výpočet v defaultním CUDA streamu, což je možné změnit pomocí funkce `cufftSetStream`, která konkrétnímu plánu nastaví stream, ve kterém se má vykonat. Pro překrytí jednotlivých výpočtů je toto nastavení potřeba udělat, to ale způsobilo, že nebylo možné použít stejný plán pro více výpočtů současně, ale pro každý stream bylo potřeba vytvořit vlastní plán, protože při současném spuštění výpočtu za použití stejného plánu v různých streamech zřejmě docházelo k souběžnému přístupu k interním alokovaným bufferům a knihovna pak nepočítala správně. Použití více plánů pak ovšem znamená potřebu více paměti na grafické kartě, protože při vytváření plánu jsou knihovou alokovány buffery pro výpočet.

6.4 Integrace do k-Wave

Využití lokální dekompozice v k-Wave bude vyžadovat některé úpravy stávající implementace pro GPU. Tyto úpravy se budou týkat zejména načítání a ukládání dat, kernelů provádějících výpočet a doplnění funkcí spojených s výměnami okrajů mezi subdoménami. Výhodou je, že se kromě pseudospektrální metody výpočet v jedné subdoméně příliš neliší od výpočtu celé domény, změny by neměly být příliš veliké a značná část implementace by měla jít znovu využít.

Protože každý uzel bude provádět výpočet jen v části domény, bude potřeba mezi jednotlivé uzly rozdělit vstupní data. Díky tomu, že k-Wave používá pro vstup soubory ve formátu HDF5, který přímo umožňuje načtení jen části datové sady, může si každý uzel načíst jen tu část pro svou subdoménu. Bude tedy nutné upravit načítání vstupu, aby byla podle čísla MPI uzlu načtena správná část dat.

Při výpočtech gradientu pseudospektrální metodou bude z důvodu rozšíření subdomény o překryv počítáno s větší datovou kostkou než vznikne dekompozicí. Proto bude nutné také upravit kernely implementující ostatní maticové operace ve výpočtu, aby tyto operace nebyly prováděny v oblasti přidaných okrajů. Provádět zbylou část výpočtu v celé rozšířené subdoméně by bylo zbytečné, protože oblast okrajů by byla přepsána daty od sousedních uzlů.

Dále bude potřeba doplnit do k-Wave samotnou implementaci dekompozice pseudospektrální metody, což zahrnuje výměny okrajů subdomén mezi uzly a doplnění do výpočtu násobení bell funkcí.

Také vzorkování a ukládání průběhu simulace, každý uzel bude vzorkovat data jen podle odpovídající části senzorové masky a opět by bylo potřeba upravit jejich ukládání jen do části datasetu ve výstupním HDF5 souboru.

Kapitola 7

Experimentální výsledky

K ověření vlastností implementace výpočtu gradientu pomocí lokální dekompozice pseudospektrální metody jsem využil superpočítač Anselm, který disponuje 23 uzly s následujícími parametry: [2]

- 2x procesor Intel Sandy Bridge E5-2470, 8 jader, 2.3GHz
- 1x NVIDIA Tesla Kepler K20
- 96 GB paměti

Hlavní parametry grafických karet NVIDIA K20 jsou:

- 5GB GDDR5 paměti
- 2496 CUDA Cores (13 MP, 192 CUDA Cores/MP)
- 2 copy engines
- PCI Express 2

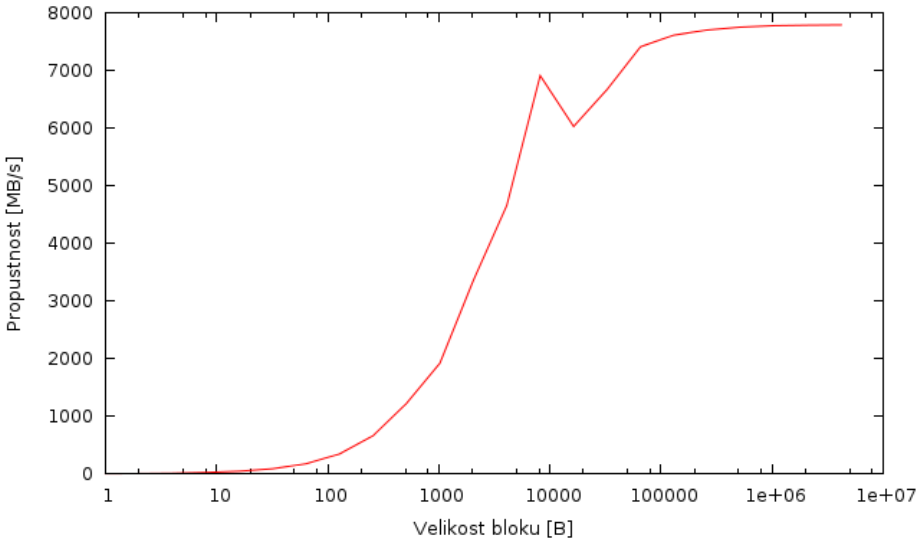
Samotná měření experimentů jsem prováděl na superpočítači Anselm, kde je jen 23 uzlů s GPU akcelerátorem, což umožnilo testování vlastností implementace při dekompozici a až mezi 16 uzlů, při rozdělení $2 \times 2 \times 4$ uzly.

Experimenty popsané v této kapitole jsou zaměřeny na efektivitu implementace této části výpočtu simulace, využití propustnosti MPI komunikace a škálovatelnost.

7.1 Propustnost MPI komunikace

Komunikace se sousedními uzly je kritická úloha, protože představuje nejvyšší režii výpočtu. Při využití 3D dekompozice si musí každý uzel vyměnit data s poměrně velkým počtem sousedních uzlů, protože k získání všech potřebných dat z okolí je nutné uvažovat nejen šest sousedních uzlů přes stěny kostky, ale i osm sousedů přes rohy a dvanáct přes hrany kostky.

Pro změření teoreticky dosažitelné propustnosti MPI komunikace jsem použil bechmark OSU [16], ze kterého jsem použil test, který měří datovou propustnost obousměrné komunikace mezi dvěma uzly pro přenos bloků různé velikosti. Výsledek měření je v grafu na obrázku 7.1, podle očekávání je pro malé bloky dat propustnost velmi nízká z důvodu režie komunikace.



Obrázek 7.1: OSU benchmark obousměrné MPI komunikace

V tabulce 7.1 jsou shrnuty výsledku měření efektivity MPI komunikace při 3D dekompozici 2x2x2 a 2x2x4 pro různé velikosti subdomény a velikosti překryvu, což jsou parametry, které určují velikost dat komunikace. Z vypočítané velikosti jednotlivých bloků okrajů a dosažené propustnosti pro danou velikost bloku dat benchmarkem OSU je vypočítána očekávaná maximální propustnost, které by bylo dobré dosáhnout. Nicméně reálně dosažená propustnost v případě 2x2x2 dekompozice dosahovala jen kolem 60 až 70 procent té maximální.

Tabulka 7.1: Dosažená propustnost MPI komunikace

rozměr subkostky	šířka překryvu	očekávaná propustnost [MB/s]	dosažená propustnost (2x2x2) [MB/s]	efektivita [%]	dosažená propustnost (2x2x4) [MB/s]	efektivita [%]
32	4	4874.34	3564.14	73.12	2961.21	60.75
32	8	6406.10	4117.68	64.28	3835.88	59.88
64	8	7211.51	4331.01	60.06	4218.42	58.50
64	16	7533.12	4595.23	61.00	4207.49	55.85
64	24	7554.83	4833.71	63.98	4216.38	55.81
128	8	7570.58	5238.70	69.20	4897.09	64.69
128	16	7719.07	5242.73	67.92	4866.19	63.04
128	24	7715.51	5202.73	67.43	4756.94	61.65
128	32	7765.44	5131.89	66.09	4601.58	59.26
256	8	7749.49	5667.90	73.14	5265.97	67.95
256	16	7772.82	5616.31	72.26	4089.87	52.62
256	24	7771.23	5550.56	71.42	5146.74	66.23
256	32	7784.49	5480.82	70.41	5059.31	64.99

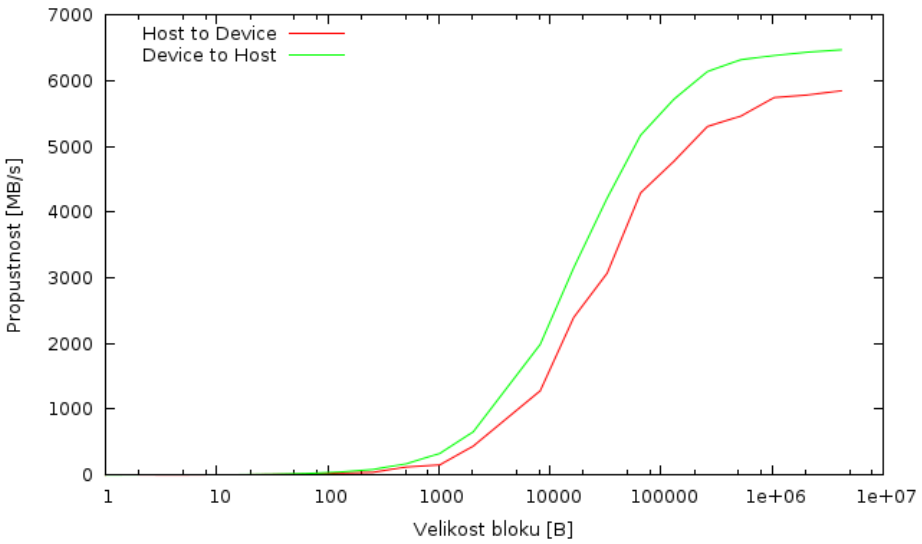
V případě dekompozice 2x2x4, kdy bylo použito 2krát více uzlů, ale také 2krát větší datová kostka, takže velikosti jednotlivých subdomén i okrajů zůstala stejná, došlo k dalšímu

poměrně značnému snížení propustnosti. To je zřejmě způsobeno větším počtem přenášených zpráv v propojovací síti, nicméně nedosažení plné propustnosti tolik nevadí, protože se podařilo komunikaci překrýt s výpočtem.

7.2 Propustnost sběrnice PCI-Express

Podobně jako v případě MPI komunikace i propustnost kopírování dat mezi paměťovým prostorem procesoru a pamětí grafické karty, která je prováděna přes sběrnici PCI-Express, závisí na velikosti bloku kopírovaných dat, jak je znázorněno v grafu na obrázku 7.2, kde je vynesena závislost propustnosti kopírování dat na grafickou kartu (Host to Device) a z grafické karty do paměti (Device to Host) na velikosti bloku. Z toho důvodu jsou data překryvů pro sousední uzly i od nich uložena v jediném lineárním bloku paměti, který je mezi hlavní pamětí a pamětí grafické karty kopírován v rámci jednoho přenosu, aby bylo dosaženo co nejvyšší propustnosti.

Potom je ovšem ještě nutné data na grafické kartě překopírovat na správné pozice uvnitř datové kostky, což znamená další režii navíc. Kopírování dat přímo na správná místa by jistě bylo také možné, ale to už by znamenalo velké množství malých přenosů, protože data okrajů v kostce nejsou v paměti spojitá, a to by mohlo být velmi neefektivní.



Obrázek 7.2: Propustnost kopírování dat mezi pamětí CPU a pamětí GPU

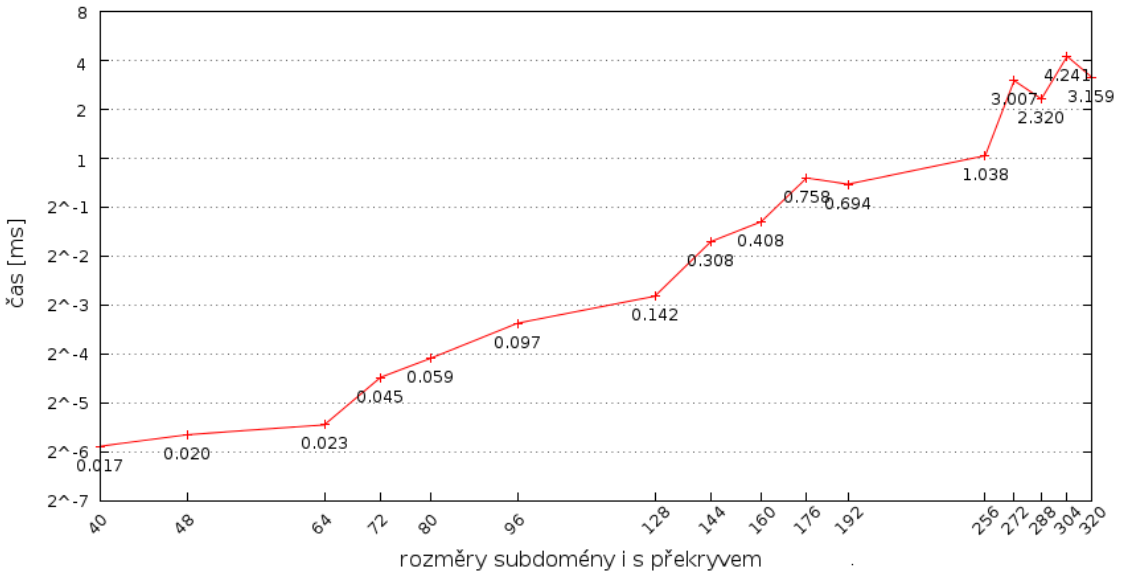
Díky tomu, že jsou data kopírována najednou, je jejich celková velikost dostatečně velká už při poměrně malých rozměrech datové kostky a šířce překryvu, aby byla dosaženo vysoké efektivity přenosu. V tabulce 7.2 je zaznamenána průměrná dosažená propustnost kopírování dat z grafické karty do paměti a naopak z paměti do na grafickou kartu pro různé velikosti subdomény z dekompozice a šířky překryvu. Z Tabulky 7.2 je patrné, že se téměř daří dosáhnout teoretické propustnosti, pouze pro malé bloky okrajů při menších velikostech datové kostky není přenos příliš efektivní, jinak jsou ale dosažené výsledky velmi dobré.

Tabulka 7.2: Propustnost mempcy

šířka překryvu	velikost subkostky	velikost okrajů [MB]	propustnost [MB/s]	efektivita [%]
4	32	0.12	4265.40	69.36
8	32	0.30	5180.98	84.24
8	64	0.95	5726.15	93.11
16	64	2.38	5901.69	95.96
24	64	4.36	5979.31	97.22
8	128	3.39	5953.61	96.81
16	128	7.62	6009.04	97.71
24	128	12.80	6013.73	97.78
32	128	19.00	6045.59	98.30
8	256	12.77	6032.16	98.08
16	256	27.12	6061.64	98.56
24	256	43.17	6051.53	98.40
32	256	61.00	6038.70	98.19

7.3 Vliv velikosti překryvu na dobu samotného výpočtu

Rozšíření subdomény o překryv neznamená jen režii spojenou s komunikací, ale už samotný výpočet trvá delší dobu, protože je celkově zvětšena velikost datové kostky. Na obrázku 7.3 je graf, který zobrazuje dobu výpočtu nad datovou krychlí s různými velikostmi, které odpovídají velikostem datových kostek o velikosti 32, 64, 128 a 256 rozšířených o překryvy o velikostech 4 až 32 bodů.



Obrázek 7.3: Doba samotného výpočtu v závislosti na velikosti dat

Z měření vyplývá, že prodloužení doby výpočtu vlivem rozšíření o překryvy je poměrně výrazné. V době výpočtu se projevuje i vlastnost implementace výpočtu rychlé Fourierovy transformace, kdy závisí na tom, jak vhodné má doména rozměry. Nejvyšší efektivita je

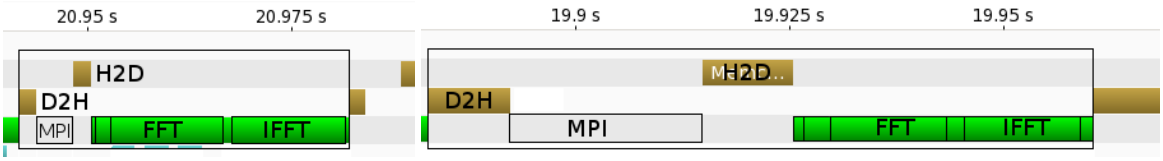
dosažena, když je velikost domény součinem malých prvočísel [12]. Nicméně i pokud celá doména před dekompozicí tuto podmínku splňuje, tak právě přidáním okrajů může být porušena.

Například pro datovou kostku o velikosti 272^3, která odpovídá subdoméně o velikosti 256^3 s okraji širokými 8 bodů, trvá výpočet 1.3krát déle než výpočet s větší kostkou 288^3, která by odpovídala stejné subdoméně s překryvem širokým 16 bodů, takže by byl výpočet nejen rychlejší, ale i přesnější. To je způsobeno právě tím, že číslo 272 je dělitelné 17, zatímco největší prvočíselný dělitel čísla 288 je 3. Pro efektivní výpočet tedy bude opravdu nutné vhodně volit velikost domény i překryvů.

7.4 Profil výpočtu

Protože použitá metoda dekompozice vede k poměrně velkému počtu komunikací a přenosů dat v paměti, bylo potřeba zjistit, kolik času je potřeba pro tuto režii vedle samotného výpočtu. K analýze, jaké poměrné zastoupení v celkové době běhu výpočtu má samotný výpočet, MPI komunikace a další režie, jsem využil nástroj NVIDIA Visual Profiler a měření doby provádění jednotlivých částí výpočtu.

Nejdříve jsem měřil profil výpočtu gradientu jen v jedné ose pro různé velikosti datové kostky po dekompozici a různé velikosti překryvů, o které je potom tato kostka ještě rozšířena, pro představu, jestli tato režie není vzhledem k výpočtu neúměrné velká. Příklad takového profilu je na obrázku 7.4, kde je vidět, jaký výrazný vliv má velikost překryvů na dobu výpočtu, protože je potřeba přenášet větší objem dat překryvů, navíc je pak datová kostka celkově větší a i samotný výpočet trvá déle. Oba tyto profily jsou z výpočtu pseudospektrální metody v subdoméně o velikosti 256^3, v prvním případě rozšířené o okraj šířky 8 bodů, ve druhém 32 bodů.



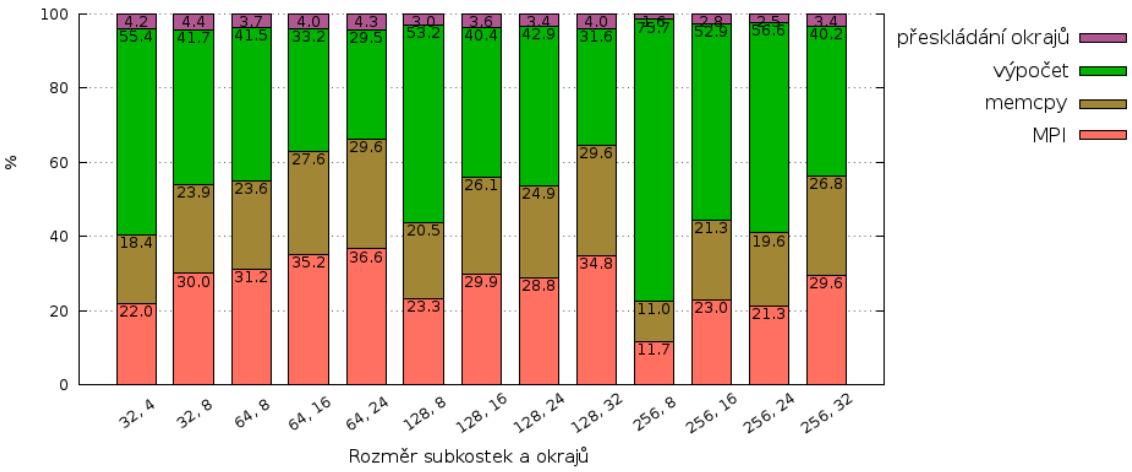
(a) 256^3, 8 (b) 256^3, 32

Obrázek 7.4: Profil aplikace bez překrytí komunikace a výpočtu

Hnědou barvou jsou v profilu zobrazeny přenosy dat z paměti grafické karty do hlavní paměti (*D2H*) i opačným směrem (*H2D*). V prodlevě mezi nimi probíhá MPI komunikace se sousedními uzly, aby se získala data překryvu. Potom ještě následuje přerovnání dat překryvů v paměti grafické karty a nakonec je samotný výpočet zobrazený zeleně, kde jeho největší část je tvořena výpočtem Fourierovy transformace.

Celkové procentuální zastoupení jednotlivých částí výpočtu v celkové době běhu je pro další rozměry datových kostek a velikostí překryvů je shrnuto v grafu na obrázku 7.5. Z tohoto grafu je zřejmé, že celková režie MPI komunikace a kopírování dat na grafickou kartu může i přesahovat 50 % celkové doby. Nejlepšího poměru se dosahuje při použití velké datové kostky a malých okrajů, velikost datové kostky je ale limitována dostupnou pamětí grafické karty a velikost okrajů zase ovlivňuje přesnost výpočtu.

Ukázalo se, že režie zabírá opravdu velkou část doby výpočtu, dalším krokem proto byla snaha překrytí této režie dalšími užitečnými operacemi a tím dosažení vyšší efektivity.

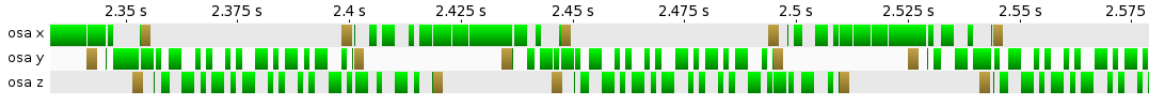


Obrázek 7.5: Profil výpočtu pro různé velikosti dat

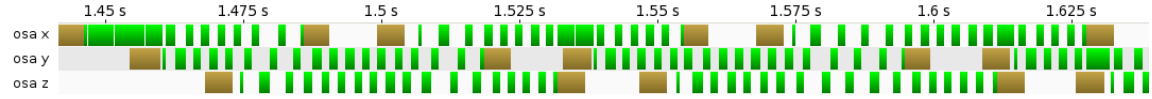
K tomuto překrytí šlo využít výpočet gradientu v dalších osách, protože jejich výpočty jsou vzájemně datově nezávislé.

7.4.1 Překrytí výpočtu a komunikací

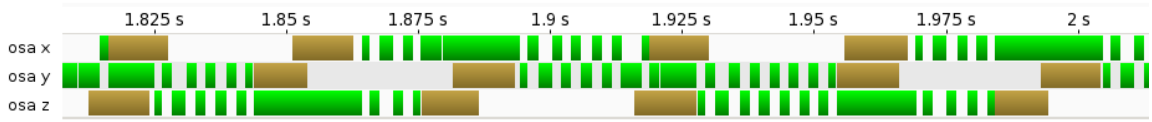
Překrytí bylo dosaženo využitím asynchronních MPI operací, asynchronních paměťových operací `cudaMemcpyAsync` a rozdělení výpočtu gradientu pseudospektrální metodou do tří CUDA streamů, kdy v každém z nich probíhá výpočet gradientu ve směru jedné osy.



(a) Velikost dat 256^3, okraje 8 bodů



(b) Velikost dat 256^3, okraje 16 bodů



(c) Velikost dat 256^3, okraje 32 bodů

Obrázek 7.6: Profil aplikace s překrytím komunikace a výpočtů v x, y a z ose

Výsledné profily jsou na obrázku 7.6, kde je zobrazeno rozdělení výpočtu ve streamech, proložení operací jednotlivých výpočtů a překrytí některých operací. V každém řádku profilu je znázorněna aktivita jednoho streamu, opět jsou hnědou barvou paměťové přenosy a

zelenou barvou výpočet, prázdná místa jsou způsobena MPI komunikací. Na profilech je vidět, jak jsou v okamžiky MPI komunikace nebo kopírování dat v jednom streamu překryty prováděním výpočtu v jiném streamu. Dokonce jsou díky dvěma kopírovacím jednotkám na grafické kartě prováděny současně i některé paměťové přenosy. Z těchto profilů to vypadá, že by se mělo podařit poměrně výrazně snížit vliv režie na dobu výpočtu. Výpočty v jednotlivých streamech jsou jsou rozděleny do menších bloků, protože je v knihovně cuFFT implementován výpočet FFT několika kernely, které byly při spouštění proloženy s výpočty z jiných streamů.

Tabulka 7.3 pak shrnuje dosaženou efektivitu, kdy byla porovnána doba samotného výpočtu bez MPI komunikace a přenosů dat na grafickou kartu se skutečnou naměřenou dobou výpočtu gradientu ve všech třech osách právě s využitím překrytí. Efektivita, vypočítaná jako podíl očekávané doby výpočtu (doba výpočtu bez režie MPI komunikace a kopírování dat mezi pamětí a grafickou kartou) s tou naměřenou, dosahuje v závislosti na velikosti dat a okrajů až 90 %, což znamená, že se překrytím podařilo výrazně snížit vliv režii až na 10 %.

Tabulka 7.3: Efektivita překrytí výpočtu a režie

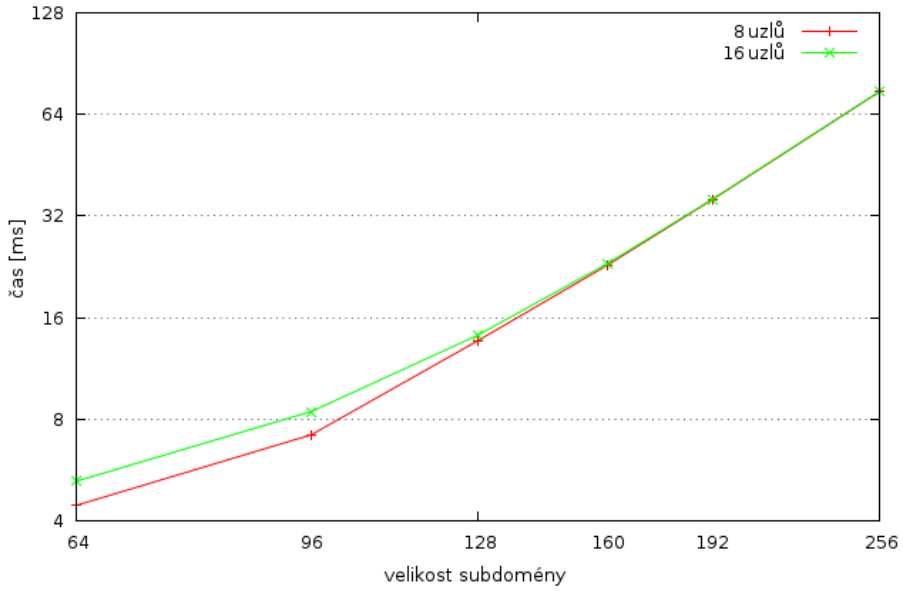
kostka	okraj	efektivita [%]
32	4	57.72
32	8	51.76
64	8	76.60
64	16	65.47
64	24	59.61
128	8	90.89
128	16	86.40
128	24	88.27
128	32	74.16
256	8	94.29
256	16	89.26
256	24	91.29
256	32	85.29

7.5 Škálovatelnost

Důležitou vlastností paralelní aplikace je její škálovatelnost, tedy vlastnosti, že při zvyšování počtu uzlů zapojených do výpočtu úměrně klesá doba výpočtu. Nebo je možné zvyšovat rozměr úlohy úměrně počtu zapojených uzlů při zachování doby výpočtu. Na superpočítači Anselm bylo možné otestovat škálovatelnost při využití až 16 uzlů.

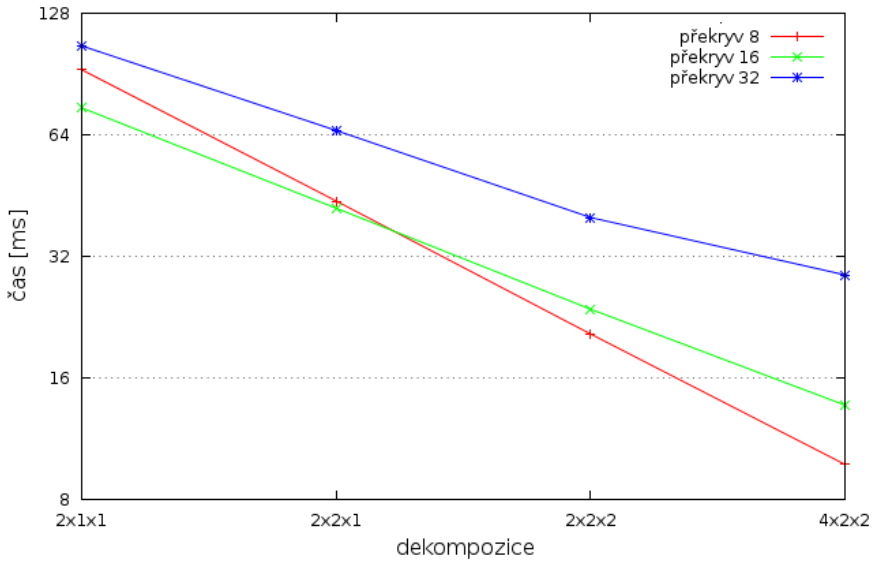
Na obrázku 7.7 je graf, který zobrazuje závislost doby výpočtu na velikosti subdomény, šířka překryvu subdomén byla zvolena vždy 16 bodů, velikost subdomény je vynesena v ose x a v ose y je vynesen čas doby výpočtu, který byl prováděn s 8 uzly a se 16 uzly. Aby byla stejná velikost subdomén v obou případech, byla doména před dekompozicí pro 16 uzlů dvakrát zvětšena. Protože jsou subdomény i vyměňované okraje v obou případech stejné, měl by i výpočet trvat stejnou dobu, což až na případy nejmenších rozměrů subdomén platí, kdy byl výpočet s 8 uzly rychlejší. Dalo by se tedy předpokládat, že i s dalším zvětšením domény i zvýšením počtu uzlů by nemuselo dojít k výraznému poklesu výkonu a mohlo by

být dosaženo stejné doby výpočtu. Bohužel to s více uzly na Anselmu není možné ověřit.



Obrázek 7.7: Škálování výpočtu při fixní velikosti subdomény pro 8 a 16 uzlů

Při měření zrychlení, kdy byla zvolena doména s pevnou velikostí $512 \times 256 \times 256$ bodů, šířka překryvů 8, 16 a 32 bodů, byl zvyšován počet uzlů od 2 po 16. Protože v prvních dvou případech není doména rozdělena ve všech osách a nejde tak o 3D dekompozici, tak jsou okraje ve směru os, ve kterých nedochází k dělení domény, přenášena jen v rámci jednoho uzlu, takže není měření příliš vypovídající. Nicméně i tak je možné sledovat trend, jak se výpočet urychlí při dalším dělení domény. Jinak by bylo opět možné použít jen 8 a 16 uzlů. Výsledek tohoto měření je v grafu na obrázku 7.8.



Obrázek 7.8: Škálování výpočtu při velikosti domény 512x256x256

Porovnání doby výpočtu při jednotlivých variantách dekompozice je shrnuto v tabulce 7.4, kde je zrychlení vypočítáno vůči dekompozici $2 \times 2 \times 2$. V případech s nejmenší šířkou překryvu dochází dokonce v každém kroku k více než dvojnásobnému zrychlení, to je ale zřejmě způsobeno vlivem velikosti domény na rychlost FFT, který je popsán v kapitole 7.3. V ostatních případech už je zrychlení nižší než 2. Z toho také vyplývá, že z hlediska ceny výpočtu (počet uzlů krát čas) bude výhodnější, pokud budou subdomény co největší, protože jejich rozdělením by nebylo zřejmě dosaženo odpovídajícího zrychlení.

Tabulka 7.4: Zrychlení výpočtu, velikost domény $512 \times 256 \times 256$

dekompozice	zrychlení (překryv 8)	zrychlení (překryv 16)	zrychlení (překryv 32)
2x1x1	1.00	1.00	1.00
2x2x1	2.11	1.77	1.62
2x2x2	4.51	3.15	2.66
4x2x2	9.50	5.45	3.69

Kapitola 8

Závěr

k-Wave již nyní obsahuje paralelní implementaci v C++, která je zaměřená na vysokou efektivitu a rychlost výpočtu simulace. Nyní se snažíme rozšířit implementaci, která využívá výpočetní výkon grafické karty, o možnost rozdělit výpočet mezi větší množství takových karet, protože výpočet na jedné kartě je velmi limitován její dostupnou pamětí. Pro dosažení dostatečné přesnosti výpočtu je potřeba hustá mřížka dat a pro větší rozměry prostoru, ve kterém se simulace počítá, může její velikost přesahovat možnosti dnešních karet.

Nevystačíme ale s dosavadním způsobem paralelizace výpočtu, tedy rozdělením výpočtu 3D FFT na 1D FFT v jednotlivých osách, protože to by znamenalo značnou režii komunikace mezi procesy při transformacích matic, navíc by kvůli využití grafických karet bylo ještě nutné provádět přenosy dat mezi hlavní pamětí a pamětí grafické karty. Použijeme proto lokální dekompozici pseudospektrální metody. Díky tomu bude výpočet probíhat na jednotlivých kartách téměř nezávisle, pouze s výměnou hodnot na okrajích jednotlivých částí rozdělené domény.

V rámci této práce byl implementován výpočet gradientu pseudospektrální metodou s využitím dekompozice ve všech třech osách k dosažení nejvyšší flexibility použití a snížení velikosti okrajů subdomén, které je nutné mezi subdoménami vyměňovat pro dosažení přesnosti. Správnost výpočtu byla ověřena oproti implementaci v MATLABu.

Experimenty na ostravském superpočítač Anselm, který kromě běžných výpočetních uzlů s vícejádrovými procesory, má také 23 uzlů, které jsou vybaveny grafickým akcelerátorem NVIDIA Tesla K20 [1]. Na tomto systému byla ověřena škálovatelnost 3rozměrné dekompozice. Vzhledem k omezenému počtu dostupných uzlů s grafickou kartou bylo využito při 3rozměrné dekompozici jen 8 a 16 uzlů. Takto bylo ověřeno, že při zvýšení počtu uzlů v dekompozici společně se zvětšením domény, aby byly zachovány rozměry jednotlivých subdomén, nedochází k poklesu výpočetního výkonu vlivem zvýšení počtu zpráv v propojovací síti uzlů. Počet těchto zpráv stoupá pouze lineárně s počtem využitých uzlů. Dalo by se tedy předpokládat, že k výraznému poklesu výkonu by nemuselo dojít ani při využití většího počtu uzlů při výpočtech ve větší doméně.

Dalším pokračováním této práce by bylo využití implementace lokální dekompozice výpočtu ve verzi k-Wave pro GPU ve výpočtu simulace šíření ultrazvuku. To by mělo umožnit provádět výpočet simulace i ve velkých doménách při zachování rychlosti výpočtu.

Při řešení práce jsem se blíže seznámil nejen s využitím grafických akcelerátorů v oblasti vysoce náročných výpočtů a způsobu práce s distribuovanými systémy, ale také se zajímavou metodou numerického řešení diferenciálních rovnic.

Literatura

[1] Anselm Cluster Documentation, Hardware Overview [online]. Dostupné na `https://docs.it4i.cz/anselm-cluster-documentation/hardware-overview`, [cit. 10. 1. 2015].

[2] Anselm Cluster Documentation, Compute Nodes [online]. Dostupné na `https://docs.it4i.cz/anselm-cluster-documentation/compute-nodes`, [cit. 18. 5. 2015].

[3] Fornberg, B.: *A practical guide to pseudospectral methods*. Cambridge University Press, 1998, ISBN 0-521-49582-2.

[4] Gabriel, E.; Fagg, G. E.; Bosilca, G.; aj.: Open MPI: Goals, Concept, and Design of a Next Generation MPI Implementation. In *Proceedings, 11th European PVM/MPI Users' Group Meeting*, Budapest, Hungary, 2004, s. 97–104.

[5] Israeli, M.; Vozovoi, L.; Averbuch, A.: Spectral Multidomain Technique with Local Fourier Basis. In *J. of Scientific Computing*, 1993, s. 135–149.

[6] Jaros, J.; Rendell, A. P.; Treeby, B. E.: Full-wave nonlinear ultrasound simulation on distributed clusters with applications in high-intensity focused ultrasound. *arXiv preprint arXiv:1408.4675*, 2014.

[7] Jaros, J.; Treeby, B. E.; Rendell, A. P.: Use of multiple GPUs on shared memory multiprocessors for ultrasound propagation simulations. In *10th Australasian Symposium on Parallel and Distributed Computing*, 2013, s. 43–52.

[8] Kirk, D. B.; Hwu, W. W.: *Programming Massively Parallel Processors. A Hands-on Approach*. Morgan Kaufmann, 2010, ISBN: 978-0-12-381472-2.

[9] Laboratory, O. R. N.: Introducing Titan [online]. Dostupné na `https://www.olcf.ornl.gov/titan/`, [cit. 24. 5. 2015].

[10] Message Passing Interface Forum: *MPI: A Message-Passing Interface Standard*. 2012.

[11] Nandapalan, N.; Jaros, J.; Rendell, A. P.: Implementation of 3D FFTs Across Multiple GPUs in Shared Memory Environments. In *Proceedings of the Thirteen International Conference on Parallel and Distributed Computing, Aplications and Technologies*, 2012, s. 167–172.

[12] NVIDIA: cuFFT Library User's Guide. Dostupné na `http://docs.nvidia.com/cuda/cufft/`, [cit. 19. 5. 2015].

[13] NVIDIA Corporation: Parallel Programming and Computing Platform. http://www.nvidia.com/object/cuda_home_new.html, 2014 [cit. 4.1.2015].

[14] Sanders, J.; Kandrot, E.: *CUDA by Example. An introduction to general-purpose GPU programming*. 2010, ISBN: 0-13-138768-5.

[15] Science and Engineering South, Centre for Innovation: Emerald [online]. Dostupné na http://www.cfi.ses.ac.uk/emerald/, [cit. 10. 1. 2015].

[16] The Ohio State University: Benchmarks. Dostupné na http://mvapich.cse.ohio-state.edu/benchmarks/, [cit. 8. 5. 2015].

[17] Treeby, B.; Cox, B.; Jaros, J.: A MATLAB toolbox for the time domain simulation of acoustic wave felds. 2012.

[18] Treeby, B. E.; Cox, B. T.: k-Wave: MATLAB toolbox for the simulation and reconstruction of photoacoustic wave-fields. 2010.

[19] Wilt, N.: *The CUDA handbook: A comprehensive guide to gpu programming*. Pearson Education, 2013.