VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

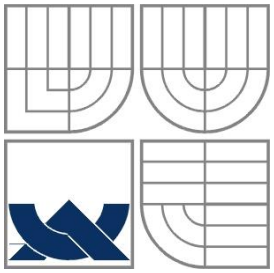
PARALELIZACE ULTRAZVUKOVÝCH SIMULACÍ S VYUŽITÍM LOKÁLNÍ FOURIEROVY DEKOMPOZICE

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

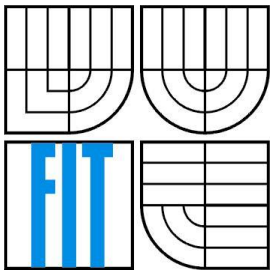
AUTOR PRÁCE
AUTHOR

BC. MATĚJ DOHNAL

BRNO 2015



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 S VYUŽITÍM LOKÁLNÍ FOURIEROVY DEKOMPOZICE

PARALLELISATION OF ULTRASOUND SIMULATIONS USING LOCAL FOURIER DECOMPOSITION

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

BC. MATĚJ DOHNAL

VEDOUCÍ PRÁCE
SUPERVISOR

ING. JIŘÍ JAROŠ, PH.D.

BRNO 2015

Abstrakt

Tato práce přináší návrh nové metody pro distribuovaný výpočet 3D Fourierovy transformace s využitím lokální 3D dekompozice domény, popis její implementace a srovnání s dosud běžně používanou metodou globální 1D dekompozice domény. Nová metoda byla navržena, implementována a testována především pro budoucí použití v simulačním programu k-Wave, ale nic nebrání jejímu použití v jiných aplikacích. Implementace prokázala svoji efektivitu na superpočítači Anselm při testování na až 2048 jádrech, kde je až 3krát rychlejší než globální 1D dekompozice za cenu nepřesnosti výpočtu v řádu 10^{-5}, neboť se podařilo významně snížit režii výpočtu v podobě komunikace mezi procesy. Na konci práce je diskutováno, jak lze s metodou výpočtu Fourierovy transformace využívající lokální dekompozici domén dosáhnout co nejlepších výsledků z hlediska přesnosti i rychlosti výpočtu, zároveň jsou zmíněny i její limity.

Abstract

This document introduces a brand new method of the 1D, 2D and 3D decomposition with the use of local Fourier basis, its implementation and comparison with the currently used global 1D domain decomposition. The new method was designed, implemented and tested primarily for future use in the simulation software called The k-Wave toolbox, but it can be applied in many other spectral methods. Compared to the global 1D domain decomposition, the Local Fourier decomposition is up to 3 times faster and more efficient thanks to lower inter-process communication, however it is a little inaccurate. The final part of the thesis discusses the limitations of the new method and also introduces best practices to use 3D Local Fourier decomposition to achieve both more speed and accuracy.

Klíčová slova

Fourierova transformace, lokální dekompozice, globální dekompozice, superpočítač, Anselm, OpenMP, MPI, pseudospektrální metoda, simulace, paralelní implementace, k-Wave toolbox

Keywords

Fourier transform, local decomposition, global decomposition, supercomputer, Anselm, OpenMP, MPI, pseudo-spectral method, simulation, parallel implementation, k-Wave toolbox

Citace

Dohnal Matěj: Paralelizace ultrazvukových simulací s využitím lokální Fourierovy dekompozice, diplomová práce, Brno, FIT VUT v Brně, 2015

Paralelizace ultrazvukových simulací s využitím lokální Fourierovy dekompozice

Prohlášení

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

.....
Bc. Matěj Dohnal
27. května 2015

Poděkování

Mé poděkování patří Ing. Jiřímu Jarošovi, Ph.D. za všechny poskytnuté rady, klidný průběh konzultací a psychickou podporu. Poděkování si zaslouží i Dr. Bradley E. Treeby, spoluautor simulačního programu k-Wave za některé další poskytnuté zdrojové kódy. Dále bych chtěl poděkovat svojí rodině za plnou podporu při psaní této práce. V poslední řadě patří poděkování organizacím IT4Innovations a MetaCentrum za poskytnuté výpočetní prostředky.

Následují povinná poděkování vyžadovaná těmito výše zmíněnými organizacemi v anglickém jazyce.

This work was supported by the IT4Innovations Centre of Excellence project (CZ.1.05/1.1.00/02.0070), funded by the European Regional Development Fund and the national budget of the Czech Republic via the Research and Development for Innovations Operational Programme, as well as Czech Ministry of Education, Youth and Sports via the project Large Research, Development and Innovations Infrastructures (LM2011033).

Access to computing and storage facilities owned by parties and projects contributing to the National Grid Infrastructure MetaCentrum, provided under the programme "Projects of Large Infrastructure for Research, Development, and Innovations" (LM2010005), is greatly appreciated.

Obsah

1 Úvod

Žijeme v době, ve které si téměř veškerá lidská činnost žádá ve všech oborech co nejlevnější, ale zároveň nejpřesnější a nejrychlejší řešení. Jedním z těchto oborů je biofyzika, která vynalezla metodu ničení nádorů pomocí ultrazvukových vln. Zásadní výhodou této metody je, že se jedná o metodu neinvazivní, tedy není potřeba jakkoli zvnějšku narušovat fyzickou celistvost pacienta. Bohužel, každá mince má i odvrácenou stranu a tou je přesné zacílení ultrazvukových vln na správné místo. Jelikož samotný impuls je velmi krátký, při špatném zacílení není možnost opravy. V tomto ohledu má výhodu chirurg, jenž může svým skalpelem kdykoli uhnout jinam.

Pro správné zacílení ultrazvukových vln tak, aby ohnisko dopadlo přesně tam, kam je požadováno, je nutné znát strukturu a vlastnosti tkáně, jíž bude ultrazvuková vlna procházet a také parametry této vlny. V ideálním případě si může medik nejdříve nanečisto odsimulovat, jak se bude určitá vlna chovat v dané tkáni a případně ještě upravit parametry ultrazvuku a pozici vysílače tak, aby bylo dosaženo lepších výsledků, a právě těmito simulacemi se zabývá program k-Wave.

S rostoucími požadavky na přesnost a zároveň rychlost simulace stávající implementace zásadních funkcí tohoto programu přestává těmto výkonnostním požadavkům dostačovat. V současné době existuje jeho verze pro superpočítačové systémy, která je schopná poskytovat přesné plány v horizontu 100 hodin, avšak pro klinickou praxi je nutné se dostat pod hranici jednoho dne [1].

1.1 Cíl práce

Cílem této práce je navrhnout distribuovanou implementaci Fourierovy transformace tak, aby nevyužívala globální, ale pouze lokální dekompozici dat, což by mělo podle předpokladů vést ke snížení objemu komunikace, navržené řešení implementovat, ověřit jeho správnost a porovnat jeho výkon oproti stávající globální dekompozici a v případě úspěchu navrhnout možné začlenění nejen do programu k-Wave. Pro dosažení tohoto cíle bude nutné zaměřit se na hledání silných a především slabých stránek současného řešení a navrhnout způsob, jak je eliminovat.

1.2 Struktura dokumentu

Práce je členěna do osmi hlavních kapitol. Po úvodní části následuje kapitola zabývající se simulačním programem k-Wave v jeho aktuální podobě a rozebírá jeho přednosti i omezení a problémy. Ve třetí kapitole jsou vyjmenovány a stručně popsány některé prostředky, které stávající implementace využívá, a nejspíše je bude využívat i nově navržená implementace, i když možná mírně odlišným způsobem.

Čtvrtá část popisuje metodu lokální dekompozice Fourierovy transformace, nejdříve odpovídá na otázku, z jakého důvodu problém řešit právě tímto způsobem, následně nastiňuje jak, a nakonec rozebírá možná omezení a problémy, kterými může tato metoda trpět. Pátá část pojednává o implementaci nově navržené metody, šestá kapitola popisuje její testování, porovnává ji se současným řešením a upozorňuje na některé faktory mající zásadní vliv na dobu výpočtu. Sedmá část diskutuje dosažené výsledky a zamýšlí se nad tím, proč by měla být nová metoda používána, proč ne, a pokud ano, jakým způsobem. Poslední část, závěr, shrnuje myšlenky prezentované v předešlých kapitolách.

Tato diplomová práce navazuje na stejnojmenný semestrální projekt, z nějž byly převzaty kapitoly číslo dvě, tři a čtyři.

2 Simulační program k-Wave

Simulační program k-Wave je sada nástrojů pro Matlab a C++, který vyvíjí pánové Dr. Bradley E. Treeby a Dr. Ben Cox z University College London a Dr. Jiří Jaroš z Vysokého učení technického v Brně jako otevřený software. Software byl vyvinut pro simulace šíření akustických a ultrazvukových vln v daném prostoru a čase. [2]

2.1 Klíčové funkce simulačního programu k-Wave

Program k-Wave používá pokročilý model simulace šíření akustických vln, který bere v úvahu akusticky heterogenní prostředí i absorpci signálu v materiálu (tkáni), umožňuje modelovat různé zdroje tlaku a rychlosti akustických vln včetně fotoakustických zdrojů nebo terapeutických či diagnostických ultrazvukových vysílačů. Rovněž uživateli nabízí možnost specifikovat detekční plochy či body, ve kterých se budou vybrané veličiny (akustický tlak, akustická rychlost částic a akustická intenzita) zaznamenávat pro další zpracování. Dále umožňuje použít původní model v reverzním režimu jako algoritmus pro rekonstrukci obrázků pro fotoakustickou tomografii. Samozřejmostí je možnost vizualizace simulace či jen měřených dat ve formě obrázků nebo videa, na uživateli je ponechána možnost, co, jak a jak často bude vizualizováno [3]. U takto odborného software rovněž nesmí chybět rozsáhlá uživatelská příručka obsahující kromě detailního popisu veškerých uživateli dostupných funkcí i několik ukázkových příkladů, na kterých je vysvětleno, jak software používat [4].

Kromě implementace celého nástroje a tedy všech funkcí v Matlabu nabízejí tvůrci i optimalizovanou verzi kritických, tedy časově velmi náročných funkcí naprogramovanou v jazyce C++ a předkompilovanou pro operační systémy Linux i Microsoft Windows (oboje pouze 64bitové architektury) pro každou generaci procesorů Intel zvlášť. Kód mimo jiné využívá pro zrychlení běhu vektorizace instrukcí procesoru, tyto techniky se ale s každou generací procesorů mění, pro dosažení maximálního výkonu je tedy nutné kompilovat program pro každou generaci procesorů zvlášť [2].

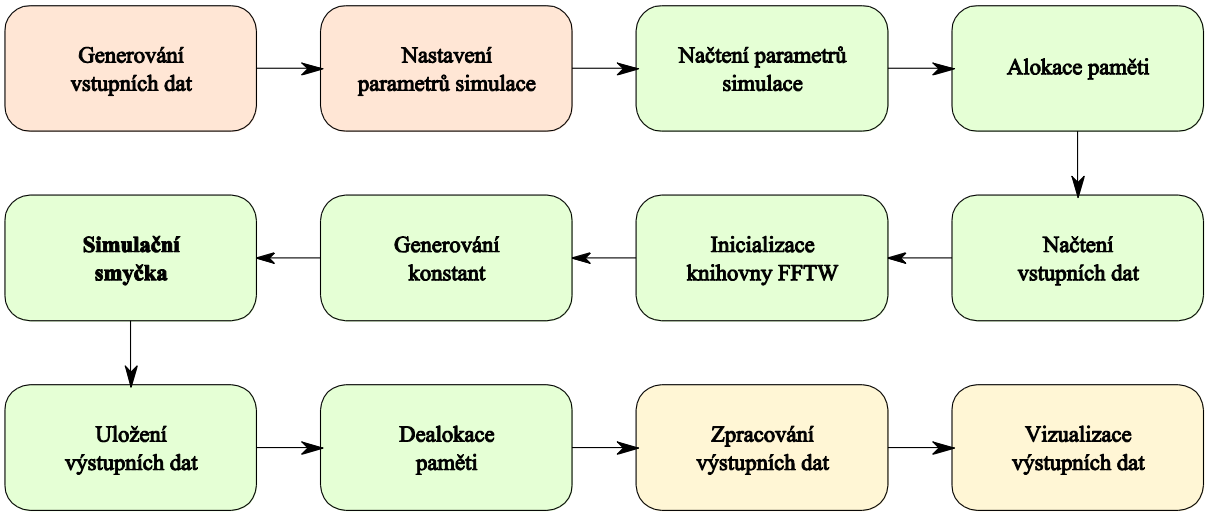
Volně stažitelnou verzi systému k-Wave je možné provozovat na běžných domácích či pracovních počítačích či serverech se sdílenou pamětí, na kterých dochází k paralelizaci výpočtu na úrovni vláken (vysokoúrovňový paralelismus OpenMP). Jádrem této práce je však distribuovaná implementace programu k-Wave pro superpočítačové systémy.

2.2 Distribuovaná implementace k-Wave pro superpočítačové systémy

Simulační software k-Wave sice lze úspěšně spouštět na běžných počítačích, ale pro rozsáhlejší simulace použitelné například v oboru medicíny již běžné počítače nenabízí dostatečný výkon, aby simulace trvala únosnou dobu. Proto je vyvíjena implementace kritických funkcí pro superpočítačové systémy, na nichž může být výpočet paralelizován mezi více uzly. V současné době existuje

implementace využívající rozhraní pro zasílání zpráv MPI[1]. Následující řádky budou věnovány popisu této stávající implementace. Tato podkapitola a i její podkapitoly vychází z [1].

Simulaci v programu k-Wave lze logicky rozdělit do třech samostatných fází, jak je znázorněno na obrázku 2.1. První fází je inicializace simulace a předzpracování dat. V této fázi se generují vstupní data, určuje se diskretizace domény na základě její fyzické velikosti a maximální sledované frekvence (čím vyšší je frekvence, tím hustější musí být mřížka, aby nedocházelo k podvzorkování), definují se materiálové vlastnosti domény a nastavují se parametry zdroje ultrazvuku. Druhou fází je samotná simulace, v níž jsou nejprve načtena vstupní data, nad těmito daty je proveden daný počet simulačních kroků a nakonec jsou výstupní data uložena. Třetí, poslední fází je analýza výsledků a jejich prezentace v člověku srozumitelné formě. Pro tuto práci je stěžejní druhá část, tedy vlastní simulace včetně čtení a zápisu vstupních a výstupních dat.



Obrázek 2.1 Orientační schéma běhu simulačního programu k-Wave. Barevné odlišení odpovídá fázím běhu programu.

2.2.1 Stávající implementace čtení a zápisu dat

K výpočtu simulace je zapotřebí uchovávat značné množství dat. Každý bod domény má definované vlastnosti (rychlost šíření zvuku, rovnovážná hustota, absorpce a další), dále se v každém bodě počítají a ukládají veličiny měnící se v čase jako akustický tlak, akustická hustota či rychlost akustické částice. Vzhledem k tomu, že některé tyto veličiny jsou vektorové, tedy mají velikost i směr, je nutné uchovávat všechny směrové složky zvlášť. Dále je potřeba mít určitý prostor k ukládání mezivýsledků. Celkově je pro výpočet potřeba 21 trojrozměrných matic o rozměrech shodných s rozměrem domény uchovávajících reálná čísla s jednoduchou přesností, dále tři matice pro uchování komplexních čísel ve Fourierově prostoru. K tomu se přidává ještě 20 jednorozměrných vektorů různých délek nesoucích mimo jiné parametry zdroje ultrazvuku a pozice bodů senzoru, kde bude docházet ke sběru výstupních dat. Nakonec je ještě potřeba zmínit dalších cca 50 hodnot, které definují velikost domény, hustotu mřížky, počet simulačních kroků a další nastavení související s během simulace.

Stávající implementace programu k-Wave, ať už ve verzi pro počítače se sdílenou pamětí nebo pro superpočítače, používá pro ukládání dat do souborů otevřený formát HDF5[2]. Pro použití v tomto

[1] MPI – Message Passing Interface, www.open-mpi.org
[2] HDF5 – Hierarchical Data Format, www.hdfgroup.org/HDF5

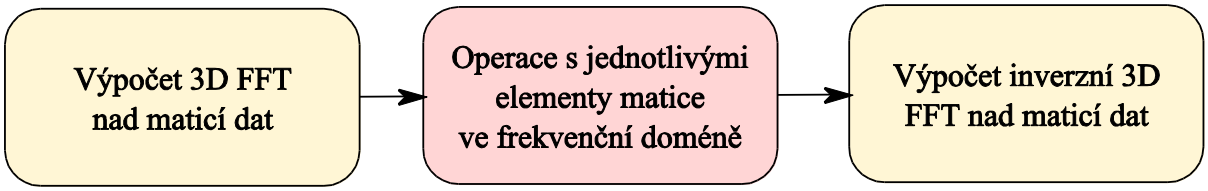
programu je stěžejní, že knihovna HDF5 je k dispozici jak ve verzi pro jazyk C++, tak i pro Matlab, dále například pro jazyk Python nebo Java a další. Knihovna HDF5 nabízí dvě rozhraní – sériové a paralelní. Sériové rozhraní se používá při generování vstupních dat. K načtení dat pro simulaci a jejich následnému uložení se používá paralelní rozhraní. Všechny procesy tedy načítají vstupní data paralelně, každý však jen tu část, která mu přísluší. Společná data jsou rovněž přečtena pouze jednou a pak rozeslána všem procesům hromadně. Podobným způsobem probíhá i zápis, tedy každý proces zapisuje svou část dat.

Data jsou v souborech vždy uložena jako trojrozměrné matice, jejichž rozměry jsou dány pomocí trojice (N_x, N_y, N_z). Skaláry, vektory a dvourozměrné matice mají nepoužité dimenze nastaveny na hodnotu 1. Pro zvýšení propustnosti dat a tedy výkonnosti jsou data v souborech rozdělena a uložena po blocích tak, aby bylo možné přistupovat k nim individuálně. Velké trojrozměrné matice jsou děleny tak, že jeden blok odpovídá jedné desce v matici, tedy trojrozměrnému poli, které má jednu z dimenzí o velikosti jedna, tudíž je vlastně pouze dvojrozměrné. Ostatní data jsou rozdělena do bloků o velikosti cca 8 MB. [1]

Navíc, ukládání po blocích umožňuje kompresi dat, bohužel však nelze komprimovaná data paralelně zapisovat, to lze pouze v sériovém režimu, ale lze komprimovaná data paralelně číst. Tato vlastnost je velmi výhodná, obsahuje-li simulovaná doména větší homogenní oblasti. Vstupní data se totiž zapisují sériově, lze je tedy zapsat v komprimované podobě a číst pak paralelně.

2.2.2 Stávající implementace simulačního kroku

Pojmem simulační krok označujeme posloupnost příkazů, které se provádí stále dokola v průběhu hlavního simulačního cyklu tolikrát, kolikrát je požadováno. V rámci simulačního kroku v programu k-Wave probíhá několikrát výpočet gradientu matice, posloupnost operací tohoto výpočtu je naznačena na obrázku 2.2.



Obrázek 2.2 Průběh výpočtu gradientu.

Výpočet Fourierovy transformace je realizován voláním funkcí z knihovny FFTW[3]. Díky faktu, že všechna vstupní data jsou reálná čísla, a výsledkem Fourierovy transformace nad tímto vstupem by bylo osově souměrné spektrum, je použita transformace typu `real_to_complex`, jejíž výstup neobsahuje redundantní data. Při výpočtu inverzní Fourierovy transformace implementace analogicky používá inverzní funkci, tedy variantu `complex_to_real`, která komplexní vstup převádí na reálný výstup.

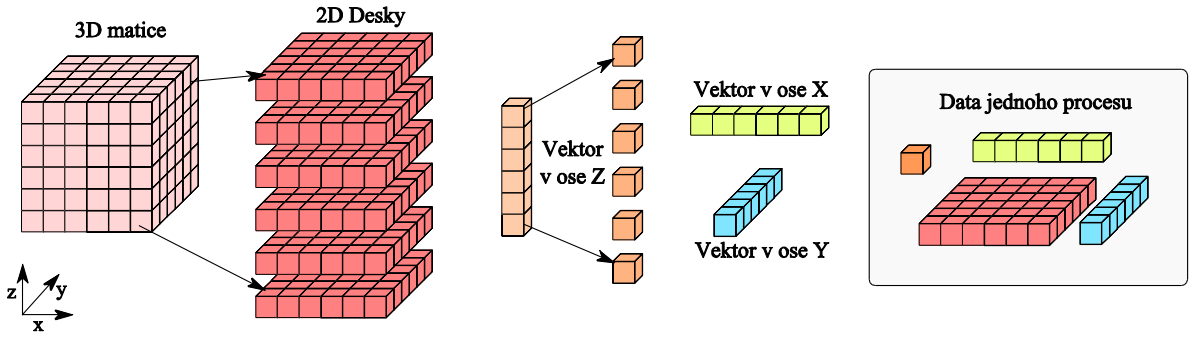
2.2.3 Dekompozice dat použitá ve stávající implementaci

Aby bylo možné dosáhnout na superpočítačovém systému co nejvyššího zrychlení, je nutné zvolit vhodnou dekompozici neboli rozdělení dat. Stávající implementace používá takzvanou globální dekompozici dat. Doména je rozdělena na desky podle osy z. Tyto desky jsou tedy opět trojrozměrné

[3] FFTW, www.fftw.org

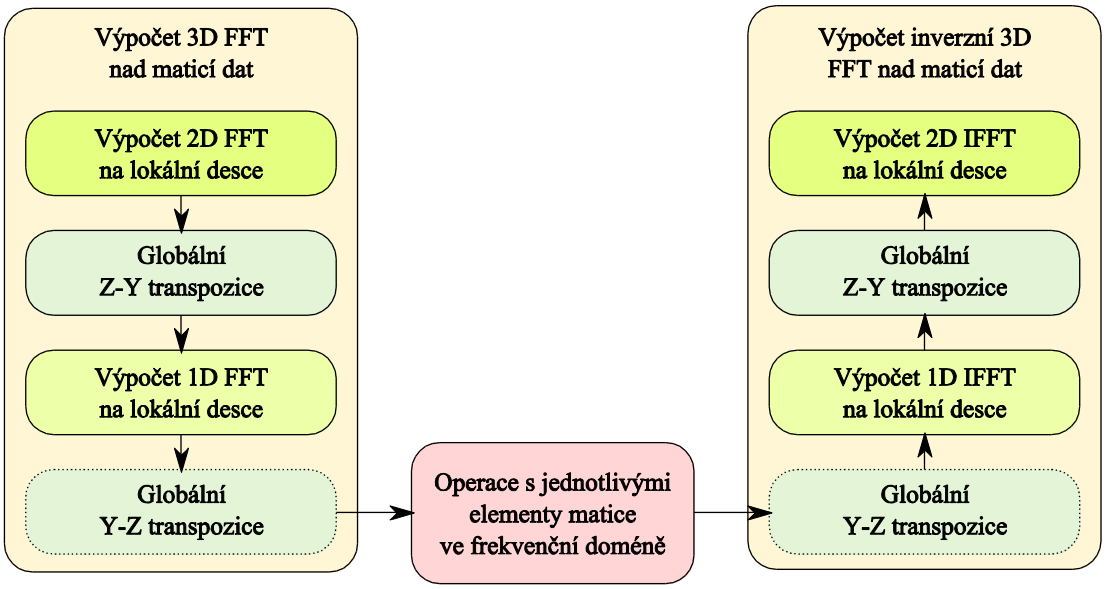
matice, jejich rozměry jsou $(N_x, N_y, N_z/P)$, kde P je počet procesů. Minimální tloušťka desky je 1, maximální N_z (původní rozměr, počítá se na 1 procesoru). Z toho vyplývá, že tato dekompozice dokáže využít maximálně $P = N_z$ procesů, případně tolik, jaký je největší rozměr domény, pokud doménu v souřadnicovém systému převrátíme tak, aby N_z byl největší rozměr.

S dekompozicí počítá už načítání dat, takže všechna jádra čtou vstupní soubor paralelně, ale každé jen svoji část dat, tedy svoji desku nebo více desek. Jak již bylo zmíněno výše, vstupní soubor je rozdělen do bloků tak, že jeden blok je právě jedna deska o rozměrech $(N_x, N_y, 1)$ (dále už jen deska). Vektory ve směru osy z jsou rovněž rozděleny a rozeslány hromadně všem procesům najednou. Vektory ve směrech os x a y a konstanty, které mají být načteny všemi procesy, jsou rovněž načteny pouze jednou a následně rozeslány pomocí `HDF5 broadcast`. Dekompozice dat je znázorněna v obrázku 2.3.



Obrázek 2.3 Globální 1D dekompozice dat použitá ve stávající implementaci.

Vlivem 1D dekompozice ale vzniká problém s výpočtem Fourierovy transformace, který ve trojrozměrné matici probíhá tak, že se nejprve vypočítá Fourierova transformace podle osy x, poté podle osy y a nakonec podle osy z, přičemž na pořadí těchto tří kroků nezáleží. Každý proces tedy vypočítá nad svými daty (2D deskou) Fourierovu transformaci podle os x a y. Poté je provedena globální Z–Y transformace matice, aby mohl každý proces vypočítat 1D Fourierovu transformaci podle původní osy z (v transponované matici podle osy y). Následně je provedena zpětná transpozice, tedy transpozice Y–Z. Pak mohou být konečně provedeny operace s jednotlivými prvky matice ve frekvenční doméně. Některé operace nevyžadují matici v původní podobě, a proto může být Y–Z transpozice vynechána. Zpětná Fourierova transformace provádí stejné kroky v opačném pořadí. Tento postup je ilustrován na obrázku 2.4, který je zpřesněním obrázku 2.2, na kterém je vyobrazen průběh simulačního kroku.



Obrázek 2.4 Průběh výpočtu gradientu s globální 1D dekompozicí dat. Operace ohraničené tečkovanou čárou nemusí být provedeny, není-li to operací mezi nimi vyžadováno.

2.3 Problémy stávající implementace pro distribuované systémy

Stávající implementace programu k-Wave pro distribuované systémy používající globální dekompozici dat bohužel trpí několika problémy, které limitují její použitelnost při větších simulacích a zároveň její rychlost výpočtu. Tyto problémy budou popsány v následujících podkapitolách, které vychází z [1].

2.3.1 Komunikace mezi procesy

Na první pohled se může zdát, že komunikace mezi procesy by mohla být jen otázkou synchronizace výpočtu, načítání vstupních dat a jejich ukládání. Při druhém pohledu však zjistíme, že největší komunikační aktivita nastává v momentě výpočtu 3D Fourierovy transformace a inverzní 3D Fourierovy transformace, konkrétně v bodě, kdy je nutné provádět Z–Y a Y–Z transpozici matice.

Předání dat mezi procesy je realizováno tak, že postupně každý proces odešle všem ostatním procesům jeden řádek o rozměru $(N_x, 1, 1)$ ze své desky. Každý z ostatních proces tento řádek přijme a uloží na novou pozici tak, aby byla provedena požadovaná transpozice (prohodí indexy y a z). Při transpozici je tedy odesláno $(N_y - 1) * (N_z - 1)$ řádků. Jeden řádek ležící na diagonále, který zůstává stejnému procesu i pro transpozici bude v dalším textu zanedbán a bude počítáno s počtem komunikací $N_y * N_z$.

Pro ilustraci objemu komunikace ukážeme na příkladu, kolik dat se při jedné transpozici matice posílá. Pro příklad stanovíme rozměr transponované matice na $1024 * 1024 * 1024$, matice obsahuje reálná čísla s jednoduchou přesností (datový typ `float`), každý prvek tedy má velikost 4 B. Lze snadno odvodit, že každý proces, kterých je 1024, odesílá všem ostatním procesům jeden řádek své desky o velikosti $1024 * 4\ B = 4\ kB$. Těchto komunikací je celkem 1024^2 (pokud proces posílá i sobě), tedy více než jeden milion.

Při zpětné Fourierově transformaci, při níž máme komplexní vstup a reálný výstup má sice matice oproti reálnému vstupu poloviční velikost (jeden rozměr se zmenšil na $\frac{N}{2}+1$), ale vstupem jsou komplexní čísla, celkový objem komunikace je ve výsledku stejný.

Dále je nutné zmínit, že tato komunikace má i určitou paměťovou režii, protože je nutné mít pro každou point-to-point komunikaci v paměti vyhrazen určitý prostor jako vyrovnávací paměť (buffer). Pro komunikaci mezi 1024 jádry může tento prostor zabírat v paměti RAM až 1 GB místa pro každé jádro, což by mohlo způsobit na některých systémech značné potíže, protože by mohla být překročena kapacita paměti. S dalším růstem počtu výpočetních jader se toto číslo bude samozřejmě dále zvyšovat.

Nakonec je nutno dodat, že veškeré komunikace mezi procesy musí být nějakým způsobem synchronizovány, což vede na další možné čekání, a tedy i zpomalení výpočtu.

Dle údajů autorů programu zabírá komunikace kolem 60 až 70 % času výpočtu.

2.3.2 Počet procesorů vůči rozměru domény

Jeden ze základních limitů stávající implementace byl okrajově zmíněn již v kapitole 2.2.3 pojednávající o dekompozici dat. Tím limitem je fakt, že počet procesů (a tedy i procesorů) může být maximálně tak velký, jak velký je největší rozměr domény. Toto omezení vyplývá z faktu, že na počítačích nelze počítat se spojitými veličinami, které jsou definovány v každém bodě. Pro výpočet je nutné tyto spojité veličiny vhodným způsobem diskretizovat. V našem případě je spojitá trojrozměrná doména převedena na trojrozměrnou matici – mřížku bodů, ve kterých jsou uloženy střední hodnoty jejich nejbližšího okolí. Pokud již máme takovouto mřížku, v dalším výpočtu již nelze její body dále dělit, a tedy proto nemůžeme počítat s deskami o tloušťce menší než jedna.

Další omezení souvisí s Z–Y a Y–Z transpozicí matice. Před transpozicí je doména rozdělena na N_z desek, každá deska přísluší jednomu procesu, po transpozici bude doména rozdělena na N_y desek. Mohou nastat tři stavy:

1. $N_y = N_z$
   V ideálním případě je nový počet desek stejný, dojde jen k přehození indexů y a z. Každý procesor počítá stejně velkou desku jako před tím.
2. $N_y < N_z$
   V tomto případě je nový počet desek menší, než byl původně. Některé procesory tedy ve Fourierově prostoru nevykonávají žádnou práci.
3. $N_y > N_z$
   Takovéto řešení by vedlo na vyšší počet procesů, než jaký je k dispozici. Program uživatele po spuštění na tuto skutečnost upozorní, v reálné doméně pak některá jádra nepracují.

V ideálním případě tedy platí, že $N_y = N_z$, přičemž doménu prvotně dělíme podle osy z. Pak může i počet procesorů být roven hodnotě N_z. V ostatních případech by měl být počet procesorů společným dělitelem čísel N_y a N_z, aby se předešlo stavu, kdy některé procesory jsou nečinné.

Fakt, že by počet procesorů měl být dělitelem čísla N_z snad již není nutné více zdůrazňovat.

3 Prostředky pro implementaci lokální dekompozice

V této kapitole budou popsány klíčové knihovny a programy nutné pro implementaci distribuovaného výpočtu Fourierovy transformace za pomoci lokální dekompozice. Rovněž bude popsán superpočítačový systém, na kterém bude implementace testována.

3.1 Superpočítač Anselm

Superpočítačový systém Anselm (dále jen Anselm) se nachází v České republice ve městě Ostrava. Byl pořízen v rámci projektu IT4Inovations[4], který je realizován mimo jiné VŠB – Technickou univerzitou Ostrava[5], Ostravskou univerzitou v Ostravě[6], Slezskou univerzitou v Opavě[7] a Vysokým učením technickým v Brně[8] a spolufinancován Evropskou unií, konkrétně z fondu regionálního rozvoje. Anselm byl uveden do provozu v květnu roku 2013.

Anselm je tvořen z celkem 209 výpočetních uzlů. Všechny uzly používají procesory Intel Xeon řady E5-2600 s architekturou Sandy Bridge. Ze všech uzlů je celkem 23 kusů vybaveno grafickým akcelerátorem NVIDIA Tesla Kepler K20, čtyři uzly jsou vybaveny koprocesorem Intel Xeon Phi 5110P a dva uzly jsou takzvané tlusté uzly, které mají osazeno 512 GB operační paměti. Zbylých 180 uzlů jsou běžné výpočetní uzly, každý se dvěma osmijádrovými procesory Intel Xeon E5-2665 a 64 GB operační paměti, na každé jádro tak připadají rovné 4 GB operační paměti. [5]

Jednotlivé výpočetní uzly jsou mezi sebou propojeny pomocí sítě Inifiniband s propustností 40 Gb/s. Topologie propojovací sítě je tlustý strom. [6]

O plánování úloh a přiřazování zdrojů se stará plánovací software PBS Professional[9].

Na superpočítači Anselm je předinstalováno a připraveno k použití velké množství rozličného software od kompilátorů a interpretů různých programovacích jazyků (python, java, c++) přes specifické výpočetní i jiné knihovny (openMP, FFTW), až po virtualizační nástroje. Mimo jiné je zde připraven i matematický software Matlab. Všechen tento software je připraven ve formě modulů, které si uživatel může kdykoli libovolně načíst do svého sezení. [7]

K superpočítači Anselm se můžeme připojit skrze terminálové rozhraní pomocí protokolu ssh nebo i s grafickým rozraním za použití rozhraní X-Window (X11). Další možností, jak používat Anselm s uživatelským rozhraním je připojení pomocí protokolu VNC. [8]

Pro práci na tomto projektu budeme na superpočítači Anselm využívat výpočetní uzly bez grafických akcelerátorů a koprocesorů.

3.2 Matlab

Matlab[10] je název pro matematický vysokoúrovňový programovací jazyk a zároveň i pro interaktivní prostředí, ve kterém se s tímto jazykem pracuje. Je vyvíjen společností MathWorks, na trh byl uveden

[4] Projekt IT4Inovations, www.it4i.cz
[5] VŠB – Technická univerzita Ostrava, www.vsb.cz
[6] Ostravská univerzita v Ostravě, www.osu.cz
[7] Slezská univerzita v Opavě, www.slu.cz
[8] Vysoké učení technické v Brně, www.vutbr.cz
[9] PBS Works, www.pbsworks.com

již v roce 1984. Aktuální verze nese název Matlab 2014a. Je určen především pro vědecké numerické výpočty a simulace. Umožňuje pohodlné zpracování a analýzu dat, stejně jako výpočty nad rozsáhlými maticemi. Obsahuje vestavěné funkce pro vizualizaci výsledků ve formě obrázků (grafů) nebo i videí. [9]

Pro programování v jazyce Matlab obsahuje vlastní editor, analýzu kódu a dokonce i profiler. Jednou z mnoha dalších funkcí je možnost na základě zadaného programu v jazyce Matlab vygenerovat program v jazyce C.

Jedná se o komerční software. V tomto projektu bude použit pro ověřování správnosti výpočtu a generování vstupních dat.

3.3 HDF5

HDF neboli Hierarchický datový formát je sada datových formátů (HDF4 a HDF5) navržených k organizaci a ukládání velkého množství numerických dat. Jeho vývoj sahá až do roku 1987, v němž byl Národnímu centru pro superpočítačové aplikace při Univerzitě v Illinois[11] zadán vývoj nové softwarové knihovny a formátu, nezávislého na architektuře, který by umožnil přenášet vědecká data mezi různými počítačovými platformami. V roce 1992 byl tento formát vybrán i pro jeden z projektů NASA, což posunulo jeho vývoj dále. Nyní je spravován a dále vyvíjen neziskovou organizací HDF Group[12]. [10]

Oficiálně jsou podporována rozhraní pro jazyky C, C++, Java, Fortran a jazyky z rodiny .NET. Existují však i rozhraní vyvíjená třetími stranami, takže lze formát HDF používat i v jazycích Python a Perl, dále pak v matematickém software Wolfram Mathematica a Matlab a mnoha dalších.

Jako klíčové vlastnosti tohoto formátu lze zmínit jeho přenositelnost a rozšiřitelnost, je vyvíjen a distribuován jako otevřený software. Formát HDF5 nijak neomezuje velikost souborů, není ani omezen na konkrétní datové typy (lze uživatelsky dodefinovat), dále umožňuje například kompresi dat.

Formát HDF5 podporuje více variant přístupu k datům, mimo jiné standardní sériový, síťový nebo paralelní. Výhoda paralelního přístupu spočívá především ve zrychlení čtení či zápisu, protože umožňuje pracovat s více datovými proudy zároveň. [11]

Knihovna HDF5 je distribuována pod BSD licencí.

3.4 FFTW

FFTW je knihovna funkcí pro jazyk C, které počítají diskrétní Fourierovu transformaci (DFT) v jedné nebo více dimenzích a libovolně velikých reálných či komplexních vstupních datech. Aktuální verze knihovny má označení 3.3.4.

Knihovna FFTW nabízí rozhraní pro jazyky C a Fortran, existují však aplikace třetích stran, které umožňují ji použít i v jiných jazycích (mimo jiné například C#, Python, Lisp, Ruby a další).

V posledních verzích podporuje vektorizaci výpočtu pomocí SSE/SSE2 nebo AVX instrukcí. Je optimalizována pro vstupní data o velikosti libovolného násobku malých prvočísel, ideálně však mocnin čísla 2.

---

[10] Matlab, www.mathworks.com/products/matlab
[11] National Center for Supercomputing Applicaticons (NCSA), www.ncsa.illinois.edu
[12] The HDF Group, www.hdfgroup.org

Jak bylo zmíněno výše, knihovna podporuje výpočet DFT nad vícerozměrnými vstupními daty, nejen nad jednorozměrnými. Dále lze při volání výpočtu DFT specifikovat, zda jsou vstupem pouze reálná či komplexní čísla.

Knihovna FFTW je distribuována pod licencí GNU-GPL verze 2.0 nebo vyšší. [12]

3.4.1 FFTW Real-to-Complex a Complex-to-Real

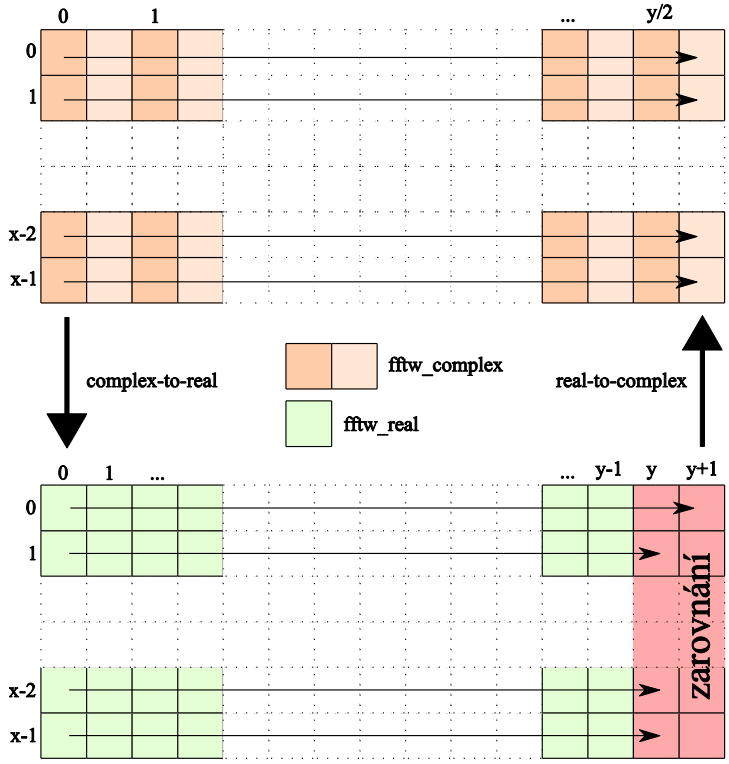
Následující podkapitola vychází z dokumentace knihovny FFTW. [13]

Provádíme-li Fourierovu transformaci nad reálným vstupem, tj. čísly s nulovou komplexní složkou, výsledné spektrum je vždy souměrné podle svého středu. Pokud tedy s jistotou víme, že vstupní data jsou pouze reálná čísla, bylo by nehospodárné uchovávat jako výsledek celé spektrum. Výsledkem transformace je pole hodnot typu `fftw_complex`, který má oproti vstupním datům dvojnásobnou velikost (ukládá zvlášť reálnou a komplexní část).

Provedením DFT real-to-complex například nad dvojrozměrným polem reálných čísel o rozměrech (N_x, N_y) vznikne pole komplexních čísel o rozměrech $(N_x, \frac{N_y}{2} + 1)$. Tuto transformaci lze používat i tak, že se jedno pole pouze přepisuje v paměti (není nutné alokovat zvlášť vstupní a výstupní), v tom případě je ale nutné počítat se zarovnáním, je-li rozměr vstupního pole sudý. V tomto příkladu by byly rozměry takového pole $(N_x, 2 * N_y + 2)$. Uvedený příklad je ilustrován na obrázku 3.1.

Provedeme-li nad takto vzniklým polem DFT complex-to-real, dostaneme zpět původní pole vynásobené počtem prvků, která je potřeba normalizovat (vydělit počtem prvků).

Výpočet DFT real-to-complex a complex-to-real samozřejmě není omezeno pouze na dvojrozměrná pole, jak bylo ukázáno na předchozím příkladu, lze ho použít na libovolný počet dimenzí.



Obrázek 3.1 Ilustrace fungování implementace knihovny FFTW při výpočtu DFT nad reálným vstupem a inverzní transformace.

3.5 GCC

GCC[13] neboli GNU Compiler Collection je sada překladačů obsahující překladače programovacích jazyků C, C++, Objective-C, Fortran, Java, Ada a Go a knihovny pro tyto jazyky. GCC bylo původně vytvořeno jako překladač pro otevřený operační systém projektu GNU. Počátek jeho vývoje se datuje do roku 1987, aktuální stabilní verze má označení 4.9.2, verze 5.0 je ve vývoji. [14] V tomto projektu budeme používat překladač GCC pro jazyk C++, lze ale použít i jiný (např. Intel).

Sada překladačů GCC je distribuována pod licencí GNU GPL verze 3.

3.6 Open MPI

Open MPI je otevřená implementace protokolu MPI neboli Message Passing Interface (rozhraní pro zasílání zpráv). Počátky vývoje sahají do roku 2004, aktuální stabilní verze má označení 1.8.4. Jedná se o rozhraní pro zasílání zpráv mezi výpočetními uzly clusteru. Mimo jiné podporuje přímou point-to-point komunikaci, hromadné zasílání zpráv všem procesům, hromadné rozptýlení dat či jejich následný sběr a další funkce. [15] Součástí je i paralelní vstupně-výstupní knihovna MPI-I/O, kterou využívá knihovna HDF5.

Open MPI je vyvíjeno komunitou The Open MPI Project jako otevřený software pod BSD licencí. Mimo Open MPI existují další implementace protokolu MPI, ať už otevřené (MPICH, LAM) nebo komerční (Intel, HP, Microsoft).

3.7 OpenMP

OpenMP[14] je specifikace množiny direktiv pro překladač, knihovních funkcí a proměnných pro specifikaci vysokoúrovňového paralelismu v programech v jazycích C/C++ a Fortran. Vývoj OpenMP začal v roce 1997, aktuální verze specifikace má označení 4.0.1. Podpora OpenMP je zabudována přímo v překladačích, pro jeho použití tak už není potřebná instalace dodatečného software nebo knihoven. [16]

Paralelizace pomocí OpenMP probíhá tak, že hlavní vlákno programu podle potřeby vytváří skupiny paralelních podvláken v místech, v nichž je to požadováno. Pomocí OpenMP se snadno paralelizují smyčky či jiné části programu, u kterých je to vhodné. Označení sekcí, které se mají v programu paralelizovat, a nastavení různých parametrů (například práce s proměnnými) se provádí pomocí direktiv `#pragma omp`. [17]

Licence OpenMP se liší podle překladače. OpenMP je použito v programu k-Wave pro běžné počítače a stroje se sdílenou pamětí, kde zajišťuje paralelizaci výpočtu.

[13] GCC, the GNU Compiler Collection, gcc.gnu.org
[14] Open Muti-Processing, www.openmp.org

4 Návrh distribuované implementace s použitím lokální dekompozice dat

V této kapitole bude nejdříve rozebrána motivace, proč by mohlo být vhodné použít lokální dekompozici dat. Dále bude rozebráno, jakým způsobem toho má být dosaženo. Nakonec budou diskutovány možné problémy, které při práci na projektu mohou vzniknout.

4.1 Motivace

V následujících podkapitolách je podrobněji rozebráno, proč se vůbec uvažuje nad lokální dekompozicí dat u systému, který s globální dekompozicí funguje. Rovněž budou vyzdvihnuty výhody lokální dekompozice oproti globální.

4.1.1 Větší variabilita počtu procesorů

Jedním z omezení stávajícího řešení je nemožnost použít více procesů (a tedy procesorů), než je nejdelší rozměr domény. Toto omezení je dáno faktem, že tloušťka desky nemůže být menší než jedna. Kdybychom chtěli desku jednoduše dále rozdělit například na poloviny, čtvrtiny či dokonce třeba jen řádky (tedy udělat 2D globální dekompozici), bylo by to možné, avšak poté už bychom museli počítat třikrát pouze 1D Fourierovu transformaci a provádět alespoň dvě transpozice matice. Objem komunikovaných dat by se sice „pouze“ zdvojnásobil, počet přenášených zpráv by se ale v jedné transpozici vynásobil tolikrát, kolik je rozměr poslední dimenze, která se dosud netransponovala. Tento počet point-to-point komunikací, který odpovídá objemu (součin všech tří rozměrů) domény by se musel navzájem synchronizovat a vytvářet pro tyto komunikace buffery. I kdyby toto řešení bylo rychlejší, mělo by mít vyšší nároky na operační paměť. Skepse ohledně rychlosti tohoto řešení pramení z faktu, že už ve stávající implementaci zabere více času komunikace mezi procesory než výpočet Fourierovy transformace.

Při hledání nového přístupu k dekompozici dat ale musíme brát ohled i na situace, kdy procesů bude méně, než je největší rozměr domény. Aby byly procesory při stávající implementaci rovnoměrně vytíženy, musí být alespoň dva rozměry domény dělitelné počtem procesorů.

Jedním z úkolů této práce bude najít algoritmus, jak rozdělit doménu pro libovolný, nebo alespoň takový počet procesorů, na který nebudou kladena takto přísná omezení.

4.1.2 Další zvětšování domény a zvyšování počtu procesorů

Již v předchozí kapitole bylo zmíněno zvýšení počtu procesorů, ovšem v kontextu dalšího dělení domény na menší části. Nyní se budeme věnovat zvýšení počtu procesorů v kontextu dalšího zvětšování domény. V tomto ohledu stávající implementace opět naráží na limity jako maximální velikost komunikačních MPI bufferů, nárůstu počtu zpráv netřeba více hovořit (při zvětšování dimenze v jednom směru roste počet zpráv lineárně, ve dvou směrech kvadraticky atd.). [1] Tímto způsobem lze doménu zvětšovat a počet procesorů navyšovat buď do momentu, kdy už nebude stačit maximální velikost MPI bufferu, nebo do okamžiku, kdy dojde operační paměť na výpočetním uzlu.

Cesta tedy musí vést jinudy, například tak, že každý proces bude komunikovat pouze svými sousedy v nejbližším okolí, těch je pouhých 26, což je oproti tisícům velmi malé číslo.

Motivací pro umožnění zvýšení počtu procesů je fakt, že v blízkosti ostravského superpočítače Anselm bude v roce 2015 uveden do provozu nový superpočítač Salomon, který bude mít 1008 výpočetních uzlů a tedy zcela jistě více než 24 000 procesorových jader doplněných 864 kartami Intel Xeon Phi. [18] Případně se nabízí použití i na jiných velkých superpočítačích.

4.1.3 Snižování paměťových a komunikačních nároků

V předchozím bodě již byly zmíněny paměťové nároky komunikace rostoucí úměrně s počtem procesorů a velikostí domény. Další motivací pro jejich snižování je existence výpočetních clusterů založených na mobilních procesorech, které mohou disponovat dostatečným výkonem, ale pouze velmi malou operační pamětí. Jako příklad takového zařízení můžeme cluster Jetson TK1, který vznikl v rámci projektu Mont-Blanc[15] v Barceloně. Jeden uzel tohoto clusteru obsahuje procesor NVIDIA Tegra K1 (4 × ARM Cortex-A15 + 1 × ARM Cortex-A7) a 2 GB operační paměti, tedy na jedno výpočetní jádro připadá pouhých 512 MB operační paměti. Dále každý uzel obsahuje grafický akcelerátor NVIDIA s architekturou Kepler. Tyto uzly jsou propojeny rozhraním Infiniband s propustností 10 Gb/s. [19]

Pokusit se rozložit data tak, aby se vešla do takto malé operační paměti je další výzvou.

4.1.4 Existující řešení

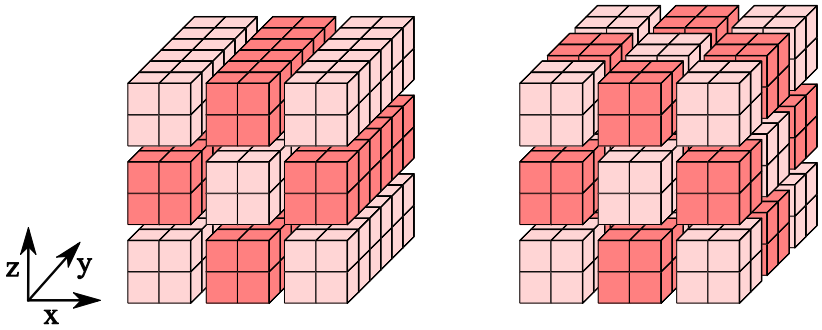
Tento bod je zmíněn právě v podkapitole motivace, protože žádné řešení řešící 3D Fourierovu transformaci pomocí 3D dekompozice nejspíše neexistuje. Existuje však několik implementací 3D Fourierovy transformace za pomoci 2D globální dekompozice, zmínit lze například knihovny 2DECOMP&FFT[16] nebo P3DFFT[17] a další.

4.2 Lokální dekompozice

Domníváme se, že problémy stávající implementace uvedené v kapitole 2.3 lze řešit pomocí jiné metody dekompozice, než je globální, tedy lokální dekompozicí. Touto metodou bychom rádi dosáhli alespoň některých cílů uvedených v kapitole 4.1, tedy snížení paměťových nároků či objemu komunikace. V následujících řádcích bude diskutováno několik variant, které se nabízejí k porovnání.

Pod pojmem lokální dekompozice si můžeme představit takovou dekompozici, při níž se celá doména rozdělí na několik menších částí (subdomén), které pak již nebudou muset komunikovat všechny navzájem, nýbrž pouze se svými nejbližšími sousedy. Fourierova transformace se při lokální dekompozici počítá pouze lokálně nad subdoménami, nikoli nad celou doménou zároveň [20].

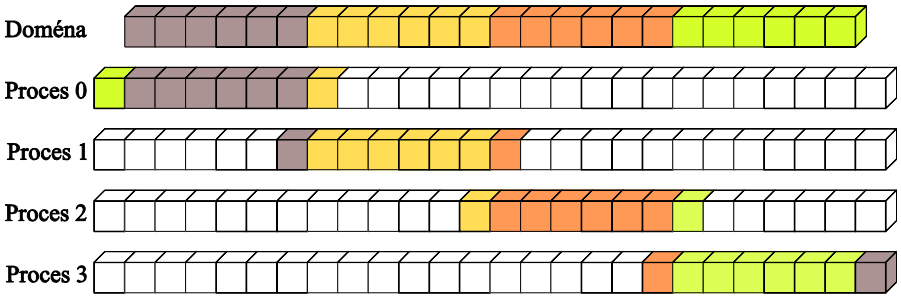
[15] Projekt Mont-Blanc, www.montblanc-project.eu
[16] 2DECOMP&FFT, www.2decomp.org
[17] P3DFFT, code.google.com/p/p3dfft



Obrázek 4.1 Některé možnosti lokální dekompozice: 2D dekompozice (řez v osách X a Z), 3D dekompozice (řez v osách X, Y, a Z). Na jeden proces připadá jeden kvádr $(2 * 2 * 6)$, respektive jedna krychle (2^3).

Tohoto dělení bychom chtěli dosáhnout tak, že nebudeme doménu dělit pouze v jedné, ale ve více dimenzích, tedy, pokud jsme dosud doménu dělili na desky, budeme ji nyní dělit například na kvádry či krychle. Řezy doménou nebudeme provádět podél jedné, nýbrž podle více os. Nad takto vzniklými subdoménami budeme již počítat 3D Fourierovu transformaci lokálně, aniž by bylo nutné z nich skládat zpět původní doménu a tu mezi procesy nějak transponovat, jak tomu je nutné ve stávající implementaci. Některé možné způsoby dekompozice ilustruje obrázek 4.1. O problémech, které tímto novým způsobem dekompozice vzniknou, pojednává kapitola 4.3.

Při pohledu na obrázek 4.1 se můžeme ptát, proč je vlastně potřeba komunikací mezi procesy? Bohužel nelze jednotlivé subdomény zcela izolovat, v čase se budou akustické vlny šířit z jedné subdomény do druhé. Bude tedy nutné řešit právě tyto komunikace mezi subdoménami. Tento problém bude řešen pomocí určitých přesahů (*overlap*) na hranicích každé subdomény. Každá subdoména tak bude mít okolo sebe obal dané tloušťky, jenž se bude aktualizovat podle okolních subdomén. Vzhledem k periodicitě FFT bude topologie propojení subdomén tvořit torus. Tím je vyřešen i potenciální problém se subdoménami na okrajích domény, kdy se do polí určených pro překryvné části zašlou data z opačné části domény. Princip rozdělení domény mezi procesy ilustruje obrázek 4.2.



Obrázek 4.2 Ukázka rozdělení jednoduché 1D domény na 4 subdomény a znázornění překryvů subdomén.

Výpočet FFT a IFFT pomocí lokální dekompozice proběhne tak, že na úplném začátku bude doména rozdělena do nějakého počtu stejných subdomén, v ideálním případě ve tvaru krychle. V každém kroku výpočtu pak proces vlastnící danou subdoménu rozešle okrajové části sousedním procesům, zároveň přijímá okrajové části od nich. Poté provede výpočet Fourierovy transformace nad takto zvětšenou subdoménou, následně operace ve Fourierově spektrální doméně a nakonec inverzní Fourierovu transformaci.

4.2.1 MPI varianta lokální dekompozice

Řešení lokální dekompozice domény pouze s pomocí MPI by bylo velmi podobné tomu, jaké je použito ve stávající implementaci, tedy doména se rozdělí na tolik subdomén, kolik máme k dispozici procesorových jader. Toto řešení povede na velký počet subdomén. S tím by mohla být spojená vyšší režie výpočtu i komunikace, protože větší počet menších subdomén implikuje větší množství překrývajících se částí.

4.2.2 Hybridní varianta lokální dekompozice

Druhou možností je kromě komunikace pomocí MPI použít i vlákna OpenMP. Tato varianta má oproti předchozímu řešení tu výhodu, že se doména dělí na méně částí, čímž logicky ubyde značná část překrývajících se oblastí a tedy i komunikace.

Doména se tak nebude dělit na tolik částí, kolik je procesorových jader, nýbrž jen na tolik částí, kolik je výpočetních uzlů, případně procesorových patic. Na každém výpočetním uzlu pak poběží pouze jeden MPI proces, který bude zajišťovat výměnu dat se sousedními procesy, pro náročné výpočty spustí paralelní OpenMP vlákna na zbylých procesorových jádrech výpočetního uzlu.

Pro porovnání uvedeme příklad srovnání této hybridní metody výpočtu lokální dekompozice a globální dekompozice. Při velikosti domény 1024^3 je pro globální dekompozici nutné v každém kroku zaslat celkem více než 1 milion zpráv o velikosti 4 kB (viz kapitola 2.3.1), tedy celkem 4 GB dat. Pro hybridní metodu předpokládejme, že máme k dispozici 16jádrové stroje, přesah jednotlivých subdomén je rovněž 16 bodů. Pak tedy dělíme doménu na pouhých $64 = 4^3$ subdomén, komunikací bude maximálně $64 * 26$ (26 je počet sousedů subdomény) a komunikovat s každým sousedem se bude maximálně 4 MB dat, celkový objem komunikace nebude větší než 1,8 GB[18] dat.

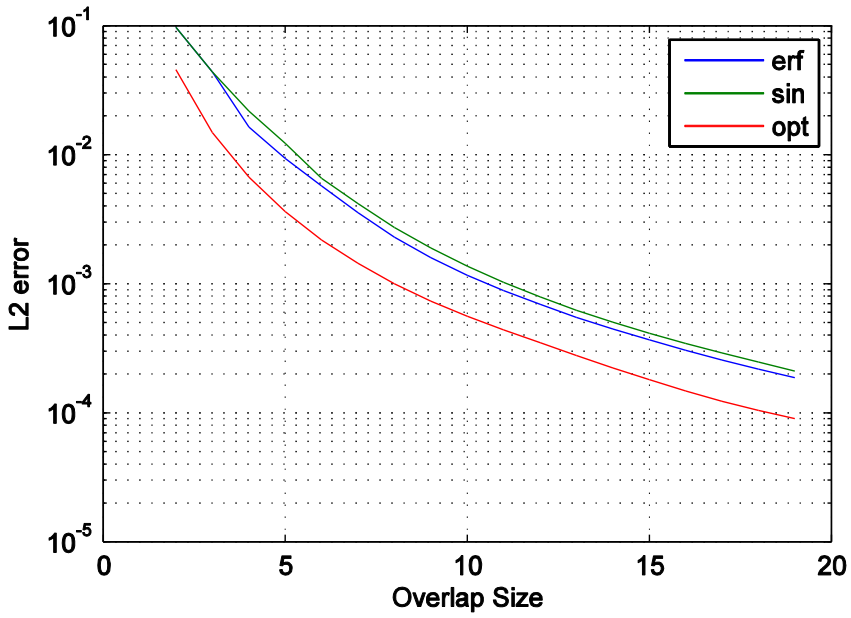
4.3 Omezení lokální dekompozice

S rozdělením domény na trojrozměrné subdomény a s počítáním Fourierovy transformace pouze nad těmito doménami se pojí několik problémů. Ty nejzávažnější budou popsány níže.

4.3.1 Nedostatečná přesnost výpočtu

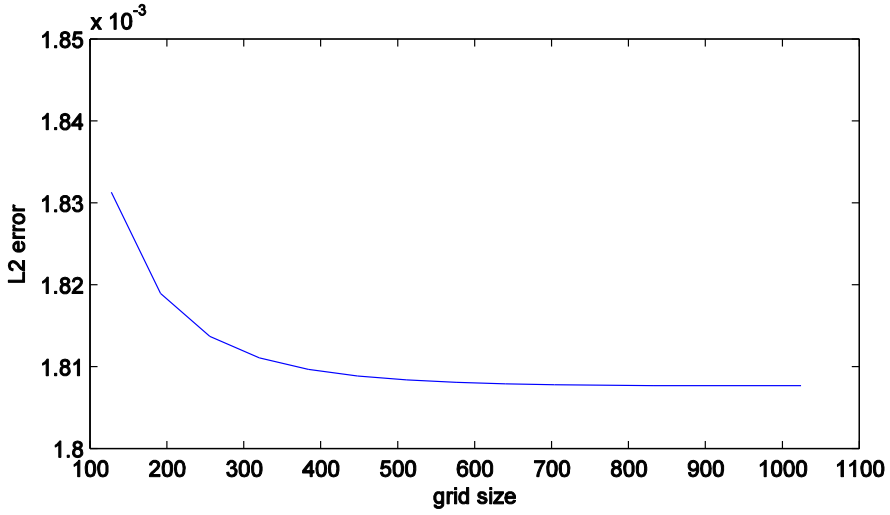
Při implementaci výpočtu Fourierovy transformace pomocí lokální dekompozice se zcela jistě setkáme s problémem, že výpočet nebude tak přesný, jako při použití globální dekompozice. Tento fakt je způsoben tím, že se Fourierova transformace nebude počítat nad celou doménou naráz, ale bude se skládat z drobnějších částí. Pokud bude doména rozdělena na moc malé části, nemusí se v takto malých samostatných částech projevit všechny frekvence a nepromítnou se do výsledku. Proto bude nutné určit takovou minimální velikost překryvu subdomén, aby celková odchylka od správných hodnot nepřekročila předem danou mez.

[18] $((256 + 2 * 16)^3 - 256^3) * 64 * 4\ B = 1\,820\,327\,936\ B \doteq 1{,}8\ GB$



Obrázek 4.3 Vliv velikosti překryvu na celkovou chybu výpočtu. 1D doména o velikosti 512, rozdělení na 2 subdomény.

Předpokládáme, že chyba se bude zmenšovat při zvětšování překryvu, jak je patrné z grafu na obrázku 4.3, který zobrazuje velikost chyby pro tři různé funkce Bell. Se zvětšováním překryvu ale naopak narůstá výpočetní režie a objem přenášených dat. Je tedy nutné najít vhodný kompromis mezi velikostí chyby a velikostí překryvu. Chyba se bude naopak mírně zmenšovat s narůstající velikostí subdomén (tedy pro přesnost výpočtu je lepším řešením menší počet větších subdomén), viz graf na obrázku 4.4.

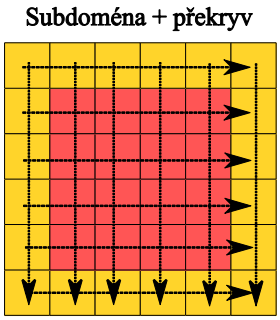


Obrázek 4.4 Vliv velikosti domény na velikost chyby výpočtu. 1D doména rozdělená na 2 subdomény, velikost překryvu: 16

4.3.2 Vysoká redundance výpočtu

Vzhledem k nutnosti překryvu určitých oblastí musíme vzít v úvahu i tímto vzniklou redundanci výpočtu, tedy fakt, že se Fourierova transformace bude pro některé (konkrétně překrývající se) části domény počítat vícekrát, než pouze jednou. Jedinou možností je počítat běžnou 3D Fourierovu

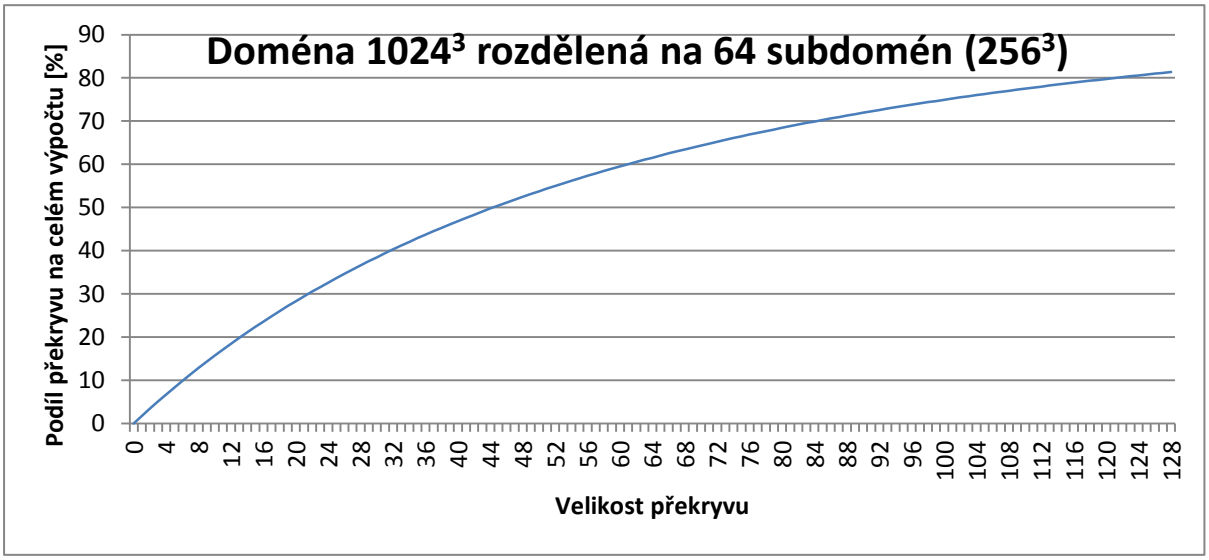
transformaci přes danou subdoménu rozšířenou o překryv tak, že celková oblast bude opět kvádr či krychle, jak je naznačeno v obrázku 4.5. Výsledek takovéto transformace bude mít stejný rozměr jako subdoména včetně překryvu.



Obrázek 4.5 Řez subdoménou (vnitřek) včetně překryvu (okraj). FFT se počítají ve směru šipek.

Za redundantní výpočet považujeme takový, kdy se Fourierova transformace počítá v nějakém bodě vícekrát, tedy veškeré výpočty v překryvných oblastech. V úvahu je nutné vzít i fakt, že některé oblasti mohou být součástí více překryvů, výpočet FFT nad nimi tedy neproběhne pouze jednou navíc, ale i vícekrát, pokaždé pro jiného souseda. Bez znalosti konkrétního rozdělení na domény na subdomény nelze specifikovat obecný vzorec pro výpočet velikosti překryvných oblastí.

Graf na obrázku 4.6 ukazuje, jaký je podíl redundantních oblastí vůči celé počítané oblasti při různé velikosti překryvu. Nutno poznamenat, že podíl větší než 50 % znamená, že suma překryvných částí je větší než celá doména a například hodnota 80 % znamená, že suma překrývajících se oblastí je čtyřikrát větší než původní doména.



Obrázek 4.6 Vliv velikosti překryvu na redundanci výpočtu.

Samostatnou otázkou zůstává, jaká bude skutečná doba výpočtu, protože knihovna FFTW používá implementaci Fourierovy transformace optimalizovanou pro domény, jejichž rozměr je roven součinu mocnin malých prvočísel, v ideálním případě pouze dvojky. Může tedy velmi záležet na rozměrech vstupní domény, počtu (tím i velikosti) subdomén a velikosti překryvu.

4.3.3 Čtení a zápis soborů, průběžné vizualizace

Nová implementace dekompozice dat by měla fungovat i se stávajícími vstupními soubory, které jsou optimalizovány pro globální dekompozici, tedy ukládány po deskách. Protože při lokální dekompozici mohou být subdomény vždy prakticky libovolně velké, nelze takto přímočaře optimalizovat strukturu souborů pro lokální dekompozici, ponecháme je tedy tak, jak jsou.

Dotazování se na hodnoty konkrétních bodů bude probíhat tak, že se ze souřadnic bodu vypočítá, ve které ze subdomén se bod nachází. Po zjištění pozice subdomény v mřížce dekompozice se zjistí číslo (rank) procesu, který subdoménu spravuje. Tento proces požadovanou hodnotu rozešle ostatním.

4.4 Omezený přístup k výpočetním prostředkům

Vývoj paralelních programů pro použití na superpočítačích má jistá specifika, jedním z nich je fakt, že prakticky nelze rozumným způsobem testovat výkonnost na běžných domácích počítačích a noteboocích. Pro ladění výkonu, ale někdy i běžné testování či hledání chyb v kódu bude nutné mít přístup k superpočítači, což vyžaduje splnění dvou podmínek: mít stabilní připojení k internetu a volnou výpočetní kapacitu superpočítače. Testování výkonnosti na větším počtu procesorových jader je tak nutné plánovat delší časovou rezervou, protože požadovaná kapacita může být často obsazena i na několik dní.

Kromě toho, že kapacita superpočítače může být obsazena jinými uživateli, může dojít ještě k situaci, že vývoj tohoto projektu vyčerpá počet jednotek (procesorových hodin), které mu byly správcem superpočítače přiděleny. Bude tedy nutné s těmito prostředky zacházet obezřetně a neplýtvat. Kromě superpočítače Anselm bude vhodné mít v záloze ještě nějaký další cluster, který bude možné použít, pokud bude primární superpočítač moc vytížený. Tuto úlohu zastane projekt MetaCentrum[19] provozovaný sdružením CESNET[20].

[19] Národní Gridová Infrastruktura MetaCentrum, www.metacentrum.cz
[20] CESNET, www.cesnet.cz

5 Implementace

V minulé kapitole bylo nastíněno, že vzniknou dvě implementační verze Fourierovy transformace využívající lokální dekompozici, jedna využívající jen a pouze zasílání zpráv MPI, druhá – hybridní – kombinující zasílání zpráv MPI s paralelizací pomocí vláken OpenMP. Základem pro obě tyto verze je první z nich, hybridní verze je pouze jejím rozšířením, proto nedříve budou popsány společné části implementace a až následně ty, ve kterých se obě varianty liší.

V průběhu implementace jsme došli k závěru, že tato práce řeší relativně obecný problém a mohla by být použita i jinde než jen pouze v simulačním programu k-Wave, metoda byla proto implementována jako třída, kterou bude možné libovolně použít v jakémkoli programu. Bude tedy nutné mít možnost srovnání se současně hojně používanou metodou globální dekompozice i mimo simulační program k-Wave. Proto byla naprogramována třetí varianta – globální dekompozice. I její implementace vychází převážně z první zmíněné varianty, proto budou opět vysvětleny pouze rozdíly oproti prvním dvěma variantám.

Samostatným implementačním problémem společným pro všechny tři varianty bylo čtení a zápis souborů ve formátu HDF5, které je společné pro všechny tři varianty.

Jako ukázka použití a zároveň pro testování byl pro každou z verzí vytvořen program počítající Laplacián (druhou derivaci podle x, y a z) vstupní matice. Posledním spustitelným programem, který vznikne překladem přiložených zdrojových souborů, je test propustnosti MPI komunikace – jedná se prakticky o totožný program jako výpočet Laplaciánu, ale dochází pouze k výměně dat a žádnému výpočtu.

Pro další podporu testování bylo vytvořeno několik krátkých skriptů pro linuxový příkazový řádek a rovněž několik programů v jazyce Matlab. I ty jsou popsány v této kapitole.

5.1 Implementace lokální dekompozice

Princip lokální dekompozice, návrh řešení i motivace, proč se jí vůbec pokoušet implementovat byl vysvětlen již v kapitole 4, níže bude rozebráno, jak toho bylo implementačně dosaženo.

Pro lokální dekompozici Fourierovy transformace je nutné rozdělit původní doménu na daný počet subdomén. Aby byl výpočet přesný a při případné simulaci se mohly tyto subdomény navzájem ovlivňovat, zadá uživatel programu velikost této překryvné oblasti. Samotný výpočet Fourierovy transformace nad takto zvětšenou subdoménou je již triviální záležitost, jedná se o běžný výpočet FFT nad 3D maticí. V této podkapitole se tedy budeme zabývat především komunikací mezi subdoménami.

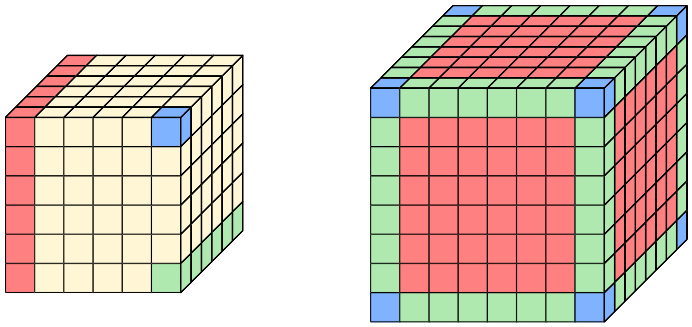
Pro uložení dat subdomény, operace nad ní a komunikaci s ostatními subdoménami byla vytvořena třída `PartialMatrix` dědící rozhraní `IMatrix`. Nejdříve ze všeho je nutné určit každému procesu, jaká je jeho pozice v mřížce dekompozice. Topologii této mřížky určuje uživatel při spuštění programu a součin délek všech dimenzí musí být roven počtu MPI procesů. Pro vytvoření mřížky je vytvořen nový MPI komunikátor – 3D kartézská mřížka, v níž má každý proces danou pozici. Podle své pozice pak každý proces načítá data ze vstupního souboru. Nikoli všechna, nýbrž pouze subdoménu o velikosti $(\frac{N_x}{P_x}, \frac{N_y}{P_y}, \frac{N_z}{P_z})$, kde N_i a $P_i, i \in x, y, z$ jsou rozměr domény v daném směru (ose) a počet procesů v mřížce v daném směru (ose). Více informací o načítání částí datových souborů najdete v kapitole 5.2.6.4.

V hlavní smyčce programu, v níž probíhá simulace, se pak opakuje následující trojice operací: nejdříve se vymění okrajové oblasti původních subdomén, poté se celá subdoména včetně okrajů vynásobí speciální maticí Bell (načítá se ze vstupních souborů), která vyhladí okraje, a konečně, jako třetí se provede Fourierova transformace nad celou doménou včetně rozšířených okrajů.

5.1.1 Definice MPI datových typů

Abychom mohli zasílat okrajové oblasti subdomény sousedním subdoménám pomocí zpráv MPI, můžeme zvolit dva přístupy. Buď můžeme data posílat vždy jako běžné 1D pole čísel typu float rozdílné délky, nebo definovat nové MPI datové typy pro určité trojrozměrné části subdomény a posílat přímo je. Vzhledem k tomu, že první varianta vy vyžadovala složité přesuny částí subdomény do pomocných polí, a zvýšila by se tak výpočetní i paměťová náročnost, bylo zvoleno druhé řešení.

Vzhledem k tomu, že se pohybujeme ve trojrozměrném prostoru a subdoména včetně okrajových oblastí má vždy tvar kvádru (či krychle), musíme z každé subdomény odesílat data 26 sousedům, stejně tak musíme data od všech 26 sousedů přijímat. To vyžaduje definici 26 nových MPI datových typů pro odesílaná data a 26 nových MPI typů pro přijímaná data, přičemž oblasti pro odesílání se překrývají, oblasti pro příjem nikoli. U každého typu (subpole) je nutné určit jeho rozměry, rozměry původního pole a pozici v původním poli, což je důvod tak velkého množství typů. Nová subpole lze rozdělit do třech základních skupin: stěny, hrany a rohy. Rozměry těchto subpolí jsou dány velikostí subdomény (bez překryvné oblasti) a velikostí překryvné oblasti. Pozice a rozměry typů pro odesílání a přijímání ilustruje obrázek.



Obrázek 5.1 Ilustrace pozice a rozměrů MPI datových typů pro odesílání (vlevo) a přijímání (vpravo) dat na doméně 6^3, překryv 1. U odesílání dat je od každého druhu (roh, hrana, stěna) vyobrazen pouze jeden zástupce.

5.1.2 Zasílání dat

Jak bylo zmíněno výše, k přenosu dat mezi subdoménami dochází vždypřed samotným výpočtem Fourierovy transformace.

Která data a v jakém množství se budou zasílat je určeno topologií kartézské mřížky (uživatel může požadovat například dělení pouze podle osy Z) a velikostí překryvu (*overlap*). Je-li doména rozdělena pouze v jednom směru, subdomény si navzájem vyměňují pouze dvě své stěny (doména má pouze dva sousedy). Při rozdělení ve dvou směrech se vyměňují stěny čtyři a rovněž čtyři hrany (doména má osm sousedů). Při 3D dekompozici se zasílá všech 6 stěn, 12 hran a 8 rohů (doména má 26 sousedů).

Všechny výše uvedené komunikace jsou implementovány jako neblokující MPI komunikace z bodu do bodu pomocí funkcí `MPI_Isend`[21] a `MPI_Irecv`[22]. Pro případ, že si dvě subdomény budou mezi sebou zasílat více než jednu položku (tato situace nastává zcela jistě při rozdělení $2 \times 2 \times 2$), musela být každému páru přiřazena unikátní značka (`TAG`), aby nedocházelo ke vzájemné záměně dat. Komunikace jsou implementovány jako neblokující, aby bylo možné v době zasílání dat provádět nějaký další výpočet nad jinými daty, čímž může být dosaženo kratšího celkového času výpočtu.

Bylo zvažováno použití hromadných sousedských komunikací `MPI_Neigbor_alltoall`[23], ale nakonec bylo od této myšlenky ze dvou důvodů upuštěno. Prvním důvodem je nutné použití nějakých dočasných polí, kam by bylo nutné data vždy přeskládat a až poté rozeslat, druhým důvodem je možná rozdílná velikost samotných komunikovaných položek – v nejhorším případě by totiž bylo možné zasílané položky slučovat pouze po dvou (stěny) a po čtyřech (hrany), pro rohy subdomény by se musela vytvořit zvláštní virtuální topologie, ve které by měl jeden bod osm sousedů.

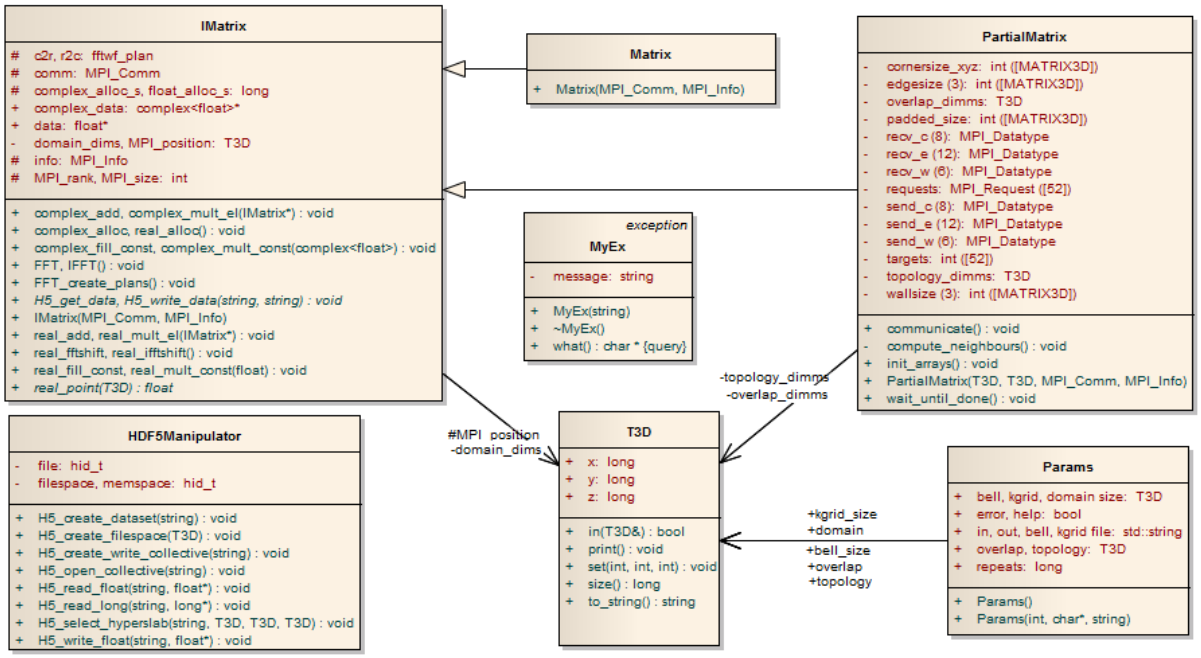
5.2 Použité třídy

Vstupním bodem celého řešení je hlavičkový soubor `Matrix.h` obsahující deklarace tříd `Marix` a `PartialMatrix`. Tyto dvě třídy implementují společné rozhraní `IMatrix`, které dědí ze stejnojmenné třídy. Hlavní odlišnost tříd `Matrix` a `PartialMatrix` je v tom, že druhá zmíněná je určena pro matice, nad kterými dané vlákno provádí výpočet ne jako nad celky, ale pouze nad jejich částí. Liší se tedy v přístupu k práci se vstupními a výstupními soubory a hlavně v třídě `PartialMatrix` je naprogramována nezbytná MPI komunikace mezi sousedními procesy (rozuměj procesy, které provádějí výpočet nad částmi matice, jež spolu sousedí). Společné metody, jako maticové sčítání, násobení (po prvku), Fourierova transformace a další jsou implementovány ve společné třídě `IMatrix`.

Samostatnou třídou je třída `HDF5Manipulator` starající se o přístup k souborům ve formátu HDF5. Třídou naprosto nezbytnou pro fungování celého řešení je malá třída `T3D`, která implementuje reprezentaci bodu v prostoru pomocí jeho souřadnic. Poslední třídou nezbytnou pro fungování popisovaného řešení je třída `MyEx`, která není nic víc, než jakýsi obal nad standardní třídou `Exception`.

Nakonec je nutné zmínit třídu `Params`, která se stará pouze o zpracování parametrů příkazové řádky a slouží až v testovacích programech, výše jmenované třídy tuto nijak nevyužívají.

[21] MPI_Isend(3) man page, https://www.open-mpi.org/doc/v1.8/man3/MPI_Isend.3.php
[22] MPI_Irecv(3) man page, http://www.open-mpi.org/doc/v1.3/man3/MPI_Irecv.3.php
[23] MPI_Neighbor_alltoall(3) man page, https://www.open-mpi.org/doc/v1.8/man3/MPI_Neighbor_alltoall.3.php



Obrázek 5.2 Generalizovaný diagram tříd, původní podoba se nachází v Příloze 5 tohoto dokumentu

5.2.1 Třída T3D

Třída `T3D` implementuje reprezentaci kartézských souřadnic v trojrozměrném prostoru, její implementace se nachází v hlavičkovém souboru `utils.h`.

5.2.1.1 Proměnné třídy

Třída `T3D` obsahuje veřejné proměnné reprezentující jednotlivé složky, tedy `x`, `y` a `z`.

5.2.1.2 Metody

Třída `T3D` má dva konstruktory, implicitní nastaví souřadnice do počátku soustavy souřadnic (bod [0,0,0]), druhý přijímá tři celočíselné parametry `x`, `y` a `z` a nastaví přímo zadané souřadnice. Souřadnice je možné nastavit i později buď přímo zápisem do proměnných, nebo naráz voláním metody `set(x, y, z)`. Metoda `size()` vrací součin čísel `x`, `y` a `z` (tedy objem kvádru, jehož strany mají délky `x`, `y` a `z`). Metoda `in()` bere jako parametr rozměry domény a říká nám, jestli aktuální bod leží v této doméně.

Metoda `to_string()` slouží k převedení souřadnic na řetězec pro snazší použití při výpisu na terminál, metoda `print()` rovnou vypisuje.

5.2.2 Třída MyEx

Třída `MyEx` obaluje standardní třídu `exception` tak, aby bylo možné při vyvolání výjimky předat vlastní řetězec s chybovým hlášením. Upraven je tedy konstruktor, který přijímá řetězec jako svůj jediný parametr a metoda `what()`, která tento řetězec vrací.

5.2.3 Třída (rozhraní) IMatrix

Třída `IMatrix` deklaruje metody a datové proměnné pro třídy `Matrix` a `PartialMatrix`, které ji dědí. Většinu z deklarovaných metod tato třída zároveň implementuje, jedná se především

o operace s maticemi. Deklarace se nachází v souboru `IMatrix.h`, implementace podle varianty v souborech `IMatrix[_varianta].cpp`.

5.2.3.1 Proměnné třídy

Třída IMatrix obsahuje dvě veřejné proměnné, jsou to ukazatele na pole dat, jeden na reálnou matici (`data`, datový typ `float`), druhý na komplexní matici (`complex_data`, datový typ `complex<float>`[24]). Tato dvě pole slouží pro uložení vstupní matice a následných mezivýsledků a výsledků. Bylo by možné je sloučit do pole jediného, jehož položky by měly datový typ `complex<float>`. Po stránce velikosti obsazené paměti jsou obě řešení stejná[25], při uložení dvou polí je z toho komplexního uložena pouze polovina (viz kapitola 3.4.1 FFTW Real-to-Complex a Complex-to-Real). Pro lepší přehlednost a možná i uživatelskou přívětivost (subjektivní) bylo zvoleno řešení používající dvě pole. Nelze nezmínit, že při použití jediného komplexního pole by se pak při MPI komunikaci musela přenášet komplexní čísla namísto reálných, i když je pro výpočet nutná pouze reálná část, nebo by se muselo používat nějaké dočasné pole.

Následuje několik chráněných proměnných, ke kterým lze přistupovat pouze ve třídách, které tuto dědí. V nich je uložen MPI komunikátor (vlastní pro každou matici) a s ním související informace jako pozice v kartézské mřížce, číslo procesu a celkový počet procesů. Dále se ukládá velikost reálné i komplexní matice pro alokaci (používá se jen při globální transpozici, kdy nemusí odpovídat skutečné velikosti) a plány pro Fourierovu transformaci.

Jedinou privátní proměnnou jsou rozměry domény `domain_dims`, ty mohou být jinými třídami pouze čteny, abychom předešli chybám přístupu do paměti.

5.2.3.2 Konstruktor a destruktor

Implicitní konstruktor (bez parametru) je chráněný, nesmí být volán z jiných tříd, nastavuje rozměr matice (domény) na nulu, inicializuje pole a FFTW plány na hodnotu `NULL`.
Veřejný konstruktor vyžaduje dva parametry, a to MPI komunikátor a MPI info. Konstruktor vytvoří duplikát komunikátoru, který mu byl předán, a uloží ho do proměnné. Každá matice má tak svůj vlastní komunikátor, při překrývání komunikace tak nemůže dojít k pomíchání dat, a ani není potřeba generovat velké množství značek (tag) pro odlišení komunikací.

Destruktor uvolňuje paměť alokovanou pro reálnou a komplexní matici (pokud proběhla alokace) a likviduje plány FFTW (pokud byly naplánovány).

5.2.3.3 Metody pro přístup k privátním proměnným

Aby bylo umožněno uživateli (programátorovi) číst a používat některé privátní proměnné, ale zároveň mu bylo zabráněno do nich přímo zapisovat, byly implementovány veřejné metody pro získávání hodnot těchto privátních proměnných. Takto lze získat velikost domény (`domain()`), velikost subdomény (`subdomain()`, včetně ohraničení `sub_padded()`), velikost matice komplexních čísel (`complex_dim()`) a další. Většina těchto metod je virtuálních, protože například výpočet velikosti subdomény se liší u matice, která je rozdělená mezi jednotlivé procesy a u takové, kterou každý proces vlastní celou.

[24] <complex>, http://www.cplusplus.com/reference/complex/
[25] float = 4 B, complex<float> = 2 * 4 B, (Matice + zarovnání) * 8 B ≈ Matice * 4 B + (Matice/2 + 1) * 8 B

5.2.3.4 Metody pro aritmetické operace nad maticí

Nad maticí (či submaticí) byly implementovány i některé základní operace. Jedná se především o sčítání dvou matic (reálných i komplexních), přičítání konstanty, násobení matic po prvcích, násobení konstantou, vyplnění konstantou. Názvy těchto metod začínají slovem `real` nebo `complex`, pokračují názvem operace (`mult`, `add`, `fill`) a liší se datovým typem parametru (`float`, `complex<float>` a `IMatrix`).

Z implementačního hlediska jde o jednoduchou smyčku či soustavu smyček for a vykonání příslušné aritmetické operace. Tyto metody jsou určeny pouze pro jednoduché operace, požaduje-li se provedení více operací nad jedním prvkem, doporučujeme pro lepší využití vyrovnávací paměti tyto operace zřetězit a provádět pouze jeden průchod maticí, použití zde uvedených metod by vedlo na více průchodů.

Speciální operací je funkce `bsxfun_times()`, která umožňuje vynásobit matici vektorem po prvcích tak, že se daný vektor rozkopíruje tak, aby vyplnil celou matici. Tato metoda má svůj vzor ve funkci bsxfun[26] pro Matlab.

5.2.3.5 Metody pro FFT

Nad maticí či submaticí třídy `IMatrix` lze samozřejmě provádět Fourierovu transformaci, k čemuž slouží metoda `FFT()`, která pouze spustí FFT real-to-complex z knihovny FFTW podle plánu, který byl před tím vytvořen. Pro zpětnou Fourierovu transformaci je tu metoda `IFFT()`, která spustí knihovní funkci pro FFT complex-to-real a ještě provede normalizaci výsledku (vydělení každé buňky celkovým počtem prvků, nad kterými byla transformace prováděna).

Vytváření plánů Fourierovy transformace se děje v metodě `FFT_create_plans()`, ve které jsou nejprve alokována pole pro FFT a následně knihovní funkce vybere nejvhodnější algoritmus výpočtu FFT. Protože proces plánování je relativně časově náročný, je možné se plány pokusit zkopírovat z jiné matice pomocí metody `FFT_copy_plans()`, což se povede, mají-li obě matice stejné rozměry. V opačném případě se místo kopírování plánů vytvoří plány nové. K načtení vstupních dat musí dojít až po provedení plánování, protože plánování by v polích načtená vstupní data přepsalo.

5.2.3.6 Metody pro práci s daty a jejich přesun

Vzhledem k tomu, že je pro uložení matic vždy potřebné mít alokovaná pole konkrétních rozměrů, byly vytvořeny metody `complex_alloc()` a `real_alloc()`, které alokaci obalují, aby se snížilo riziko vzniku chyby zadáním nesprávné velikosti pole. Protože uživatel může potřebovat načíst matici reálných čísel (třída `HDF5Manipulator` pracuje pouze s reálnými čísly) a tu pak převést na komplexní, byla implementována metoda `copy_real_to_complex()`, která zkopíruje reálnou matici (resp. její polovinu) v poli `data` do pole `complex_data`.

Dále byly implementovány metody `real_fftshift()` a `real_ifftshift()`, které, stejně jako jejich vzory v Matlabu[27], provádějí přehození poloprostorů podle os x, y a z. Při této operaci je vytvořeno dočasné pole, do nějž jsou vloženy odpovídající prvky z pole původního po přepočítání indexů. Funkce FFTShift nebo IFFTShift se zpravidla provádí nad maticí pouze jednou, doba jejich provádění proto není kritická, ale jistě by bylo možné nalézt efektivnější řešení. Tyto dvě metody nejsou implementovány v globální verzi dekompozice.

[26] MATLAB bsxfun, http://www.mathworks.com/help/matlab/ref/bsxfun.html
[27] MATLAB fftshift, http://www.mathworks.com/help/matlab/ref/fftshift.html

5.2.3.7 Virtuální a ostatní metody

Speciální metodou ve třídě `IMatrix` je metoda `MPI_barrier()`, která obaluje volání stejnojmenné MPI funkce nad tou kopií komunikátoru, kterou používá daná matice. Tuto metodu lze použít například pro synchronizaci potenciálních kontrolních výpisů při výpočtu.

Nakonec musíme zmínit ještě tři metody, které jsou čistě virtuální a každá ze tříd, které tuto třídu dědí, si je musí implementovat samy. Jde o metody pro čtení a zápis datasetu z/do HDF5 souboru s názvy `H5_get_data()` a `H5_write_data()` a metodu pro získání hodnoty bodu dle zadaných souřadnic `real_point()`, jejíž implementace bude v nerozdělené matici triviální, ale v rozdělené vyžaduje bližší popis.

5.2.4 Třída Matrix

`Matrix` je implementací třídy pro takovou matici, kterou má proces načtenou celou, nikoli pouze její část, konkrétně tedy slouží pro uložení matic KGrid a Bell. Třída `Matrix` dědí třídu `IMatrix` a navíc implementuje metody pro čtení a zápis datasetu z/do HDF5 souboru. Deklarace třídy se nachází v souboru `Matrix.h`, implementace v souboru `Matrix.cpp`.

5.2.4.1 Metody pro přístup k HDF5 souborům

Metody `H5_get_data()` a `H5_write_data()` obalují volání metod třídy HDFManipulator tak, aby daný proces vždy četl nebo zapisoval celou matici ze zvoleného datasetu. Vstupní i výstupní soubory jsou otevírány kolektivně, lze k nim tedy přistupovat ze všech procesů zároveň. Načtená data ze souboru se uloží do pole `data` (pokud se podaří ho alokovat), matice obsažená v tomto poli se rovněž do souboru zapisuje. Obě metody vyžadují právě dva parametry: cestu k souboru a cestu k datasetu v rámci souboru.

5.2.5 Třída PartialMatrix

`PartialMatrix` je implementací třídy pro uložení takové matice, ze které má každý proces uloženou pouze část, tedy v kontextu celé doménu má pouze jednu subdoménu plus překrývající se části s jinými subdoménami (overlap). Třída `PartialMatrix` dědí třídu `IMatrix`, přičemž některé její proměnné předefinuje. Navíc přidává metody pro komunikaci mezi procesy, aby si mohly vyměňovat okrajové části subdomén. Deklarace třídy je v souboru `Matrix.h`, implementována je v souboru `PartialMatrix.cpp`.

5.2.5.1 Konstruktor

Konstruktor třídy `PartialMatrix` inicializuje některé privátní proměnné (pole, FFTW plány) na hodnotu `NULL`, vytváří nový MPI komunikátor – kartézskou mřížku, konkrétně tedy 3D torus a získává a ukládá své vlastní souřadnice v rámci této mřížky.

5.2.5.2 Privátní proměnné

Jelikož třída `PartialMatrix` implementuje výměnu dat mezi procesy, přidává oproti běžné matici třídy `Matrix` značné množství privátních proměnných spojených právě s MPI komunikací. Jsou to především proměnné typu `MPI_Datatype` pro uložení nových MPI datových typů, kterými vymezíme v rámci subdomény ty oblasti, které budeme posílat a přijímat.

Dále je zde pole `requests` pro uložení položek typu `MPI_Request`, ve kterém pak budeme kontrolovat, zda již proběhly všechny požadované komunikace. Dalším polem stejné velikosti je pole

`targets`, kde jsou v příslušných položkách pole uloženy cíle či zdroje jednotlivých komunikací (komu posíláme, od koho přijímáme) ve formě čísla cílového procesu.

Nakonec jsou zde i pole pro uložení vypočtené velikosti jednotlivých posílaných položek, tedy rozměry stěn, hran a rohů.

5.2.5.3 Výpočet velikosti a inicializace MPI datových typů

Dříve, než mohou procesy začít mezi sebou vyměňovat okrajové části svých subdomén, je nutné inicializovat nové MPI datové typy pro posílání stěn, hran a rohů. Tento výpočet probíhá v metodě `init_arrays()`. Nejdříve se vypočítají velikosti posílaných oblastí, které se uloží do pomocných polí. Následně jsou pomocí funkce `MPI_Type_create_subarray()` vytvářena nová pole pro MPI komunikaci mezi procesy. Jako parametry do této funkce vstupují: počet dimenzí, rozměr původního pole, rozměr nového pole, odsazení od počátku soustavy souřadnic a způsob uložení prvků v paměti. Mimo velikosti polí je tak potřeba pro každou položku zvlášť vypočítat odsazení od bodu [0,0,0].

Nové datové typy (či pole, chcete-li) jsou vytvářeny pouze ty, které jsou pro aktuální spuštění programu potřebné – zvolil-li uživatel dekompozici například pouze podle osy Z, vůbec se nevytvářejí pole pro hrany a rohy, stejně tak pole pro stěny při dekompozici podle os X a Y. Pro zjednodušení by mohly být vytvářeny, ale bohužel nelze vytvořit datový typ pole, jehož jedna délka je nulová, a tedy celková velikost pole je nulová. Na fakt, že jsou vytvářena jen některá pole, a ne všechna, je nutné brát zřetel při rušení MPI datových typů a polí ve funkci `destroy_arrays()`.

S inicializací polí úzce souvisí i vyhledání cílů jednotlivých komunikací, kdy proces zjišťuje MPI rank každého ze svých 26 sousedů a ukládá si tato čísla do pole `targets`, což zajišťuje metoda `compute_neighbours()`.

5.2.5.4 Metody pro MPI komunikaci mezi sousedními procesy

Stěžejní metodou pro komunikaci a výměnu dat mezi procesy je metoda `communicate()`, která zahájí odesílání dat ostatním procesům a zároveň také zahájí čekání na přijetí dat od ostatních procesů. Stejně jako při vytváření inicializaci polí pro zasílání dat ani zde nejsou posílána žádná zbytečná data, ostatně ani není kam, protože pro ně nejsou vytvořena pole. Lze tak opět posílat například pouze dvě stěny, či pouze čtyři stěny a čtyři hrany.

Samotné posílání a přijímání dat je realizováno funkcemi `MPI_Isend` a `MPI_Irecv`. Na konci metody `communicate()` se nečeká na dokončení přenosu dat, aby mohlo být možné komunikaci překrýt nějakým užitečným výpočtem. Čekání na dokončení komunikace se provede voláním metody `wait_until_done()`, která zastaví program do doby, než všechny komunikace zaregistrované v poli `requests` doběhnou do konce.

Nakonec je nutné zmínit i metodu `communicate_blocking()`, která obaluje volání dvou výše zmíněných, čímž svým způsobem simuluje blokující přenos, ale pořád se nejedná přímo o použití blokujících funkcí `MPI_send` a `MPI_recv`.

5.2.5.5 Metody pro přístup k HDF5 souborům

Implementace metod `H5_get_data()` a `H5_write_data()` je téměř stejná jako ve třídě Matrix, ale nyní není čtena matice celá, nýbrž jen její část podle zvolené topologie dělení domény a pozice procesu v kartézské mřížce. Celá implementace se tak liší pouze voláním metody `HDF5Manipulator::H5_select_hyperslab` s patřičnými parametry.

5.2.5.6 Metoda pro čtení hodnoty konkrétního bodu domény

Uživatel programu může chtít číst jeden (či více) konkrétní bod, jehož pozici zadá pomocí jeho souřadnic. Tuto funkčnost implementuje metoda `real_point()`, jejímž parametrem jsou právě souřadnice bodu, návratovou hodnotou hodnota uložená právě v tomto bodu. Zdánlivě jednoduchý implementační úkol naráží na fakt, že každý proces vlastní pouze část celé matice a aby všechny procesy dokázaly vrátit korektní hodnotu, je nejprve nutné najít konkrétní subdoménu, ve které se bod nachází a poté jeho hodnotu pomocí funkce `MPI_Bcast()` rozeslat všem ostatním.

5.2.6 Třída HDF5Manipulator

Třída `HDF5manipulator` obsluhuje čtení a zápis souborů ve formátu HDF5, v nichž jsou uložena vstupní data a do nichž se ukládají výstupní data. Deklarace je uvedena v souboru `HDF5Manipulator.h`, definice v souboru `HDF5Manipulator.cpp`. Metody této třídy obalují některé z funkcí paralelní verze knihovny HDF5 tak, aby bylo uživateli usnadněno jejich použití, především aby nemusel vždy nastavovat některé příznaky a případně počítat pozice submatic v souboru. Pokud by uživateli-programátorovi některé z funkcí chyběly, může nadále přímo používat funkce z knihovny HDF5.

Níže je uveden popis metod a proměnných třídy `HDF5Manipulator`, pro stručnost jsou podobné metody sloučeny do jedné podkapitoly.

5.2.6.1 Proměnné třídy

Třída `HDF5Manipulator` obsahuje následující proměnné:

- `file` – ukládá popisovač otevřeného souboru
- `filespace` – ukládá informaci o pozici dat v HDF5 souboru
- `memspace` – ukládá informaci o rozložení dat v paměti počítače
- `comm`, `info` – MPI komunikátor

Všechny proměnné třídy `HDF5Manipulator` jsou v této třídě privátní a nelze k nim zvenčí přistupovat jinak, než přes metody této třídy.

5.2.6.2 Otevření a zavření souboru

Při otevírání souboru rozlišujeme, zda soubor otevíráme pro čtení nebo zápis. Pokud se nepodaří soubor otevřít, je vyvolána výjimka třídy MyEx obsahující hlášení příslušné této chybě.

Otevření souboru pro čtení zajišťují metody `H5_open_single()` a `H5_open_collective()`, zápis `H5_write_single()` a `H5_write_collective()`, jediným parametrem těchto metod je cesta k souboru předaná jako řetězec. Otevření souboru pro zápis způsobí ztrátu již existujícího souboru. Metody obsahující v názvu slovo „single“ jsou určeny především pro ty případy použití, ve kterých každý proces čte/zapisuje z/do jiného souboru, případně čte/zapisuje pouze jeden proces. Pokud je metoda vykonána úspěšně, je daný soubor otevřen a jeho popisovač uložen v privátní proměnné.

Zavření souboru se provádí voláním metody `H5_close()`, která korektně zavře soubor a smaže popisovač.

5.2.6.3 Čtení a zápis dat

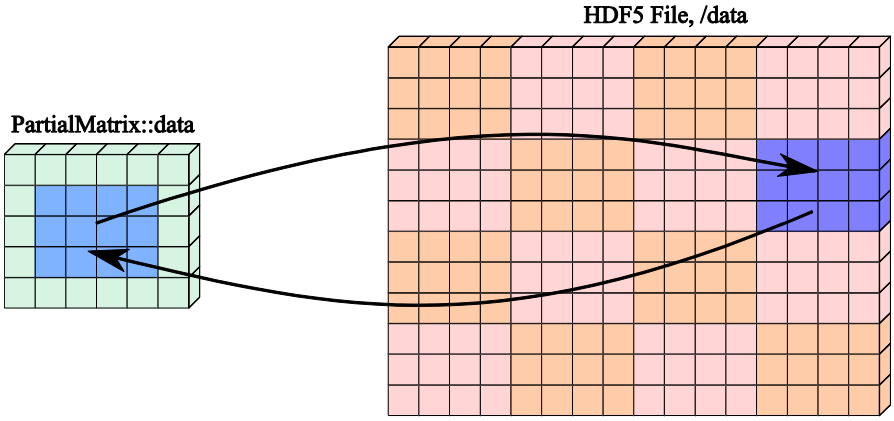
Pro čtení a zápis dat jsou připraveny metody `H5_read_long()`, `H5_read_float()`, `H5_write_float()` a `H5_write_float_single()`, které požadují právě dva vstupní

parametry – lokaci dat (cestu k datasetu v rámci souboru) a odkaz na pole v paměti, odkud se budou data zapisovat do souboru, či kam se budou číst. Jako vstupní či výstupní soubor je použit ten otevřený soubor, jehož popisovač je uložen v privátní proměnné `file`. Pokud chceme číst/zapisovat jen část datasetu či použít jen určitou část pole, je nutné před čtením nebo zápisem volat metodu `H5_select_hyperslab()`. V implicitním nastavení je použit celý dataset a celé datové pole.

5.2.6.4 Výběr oblasti dat v rámci datasetu i paměti

Aby bylo možné načítat či zapisovat z matice každým MPI procesem jen určitou část dat, je nutné tuto část nějak vymezit. Stejně tak může být velmi přínosné určit například odsazení čtených/zapisovaných dat v rámci pole. Tyto dvě informace se nastavují naráz pomocí metody `H5_select_hyperslab()`, která přijímá jako parametry (i) cestu k datasetu (ii) pozici MPI procesu v rámci kartézské mřížky (iii) rozměry submatice a (iv) rozměry submatice včetně překryvů či odsazení. Metoda vypočítá vše potřebné a informace o pozici dat v souboru a rozložení v paměti uloží do privátních proměnných `filespace` a `memspace`. Pokud některá z operací selže, je vyvolána výjimka `MyEx` s odpovídajícím chybovým hlášením.

Po použití je vhodné zvolený datový prostor zavřít voláním funkce `H5_close_hyperslab()`, která nemá žádné parametry (operuje nad privátními proměnnými).



Obrázek 5.3 Ukázka použití metody H5_select_hyperslab, doména [16,12,1], topologie [4,4,1], překryv 1. Odpovídající volání: `H5_select_hyperslab("/data", T3D(2,3,0), T3D(6,5,1), T3D(4,3,1))`

5.2.6.5 Zjednodušené metody pro čtení a zápis

Pro absolutní jednoduchost použití byly navíc implementovány metody takové, aby se dal provést zápis či čtení napsáním jednoho řádku kódu. Tyto metody pouze obalují výše zmíněné metody.

Zjednodušená metoda pro zápis má tu nevýhodu, že pomocí ní lze zapsat do souboru pouze právě jeden dataset. Její název je rovněž `H5_write_float()`, ale musí být volána právě se čtyřmi parametry, z nichž první je cesta k souboru, druhý cesta k datasetu v rámci souboru, třetí je rozměr zapisované matice a čtvrtý ukazatel na data. Výše zmíněná nevýhoda této metody je způsobena tím, že daný soubor je ihned po zápisu zavřen. Dalším otevřením by pak byl vymazán.

Metoda `H5_read_float()` je volána se třemi parametry, první je cesta k souboru, druhý cesta k datům v rámci souboru a třetím ukazatel na pole dat, kam mají být data ze souboru načtena.

Poslední metoda `H5_get_3D()` bere jako vstupní parametry cestu k souboru a tři cesty k datasetům (jejich názvy), vrací objekt třídy `T3D`. Obaluje tři volání `H5_read_long()` nad polem o velikosti jedna. Slouží k načtení souřadnic, například rozměru domény.

5.2.7 Třída Params

Třída Params slouží ke zpracování a uchování argumentů příkazové řádky. Třída je implementována v souboru `Params.h`. Objekt této třídy uchovává jaké své veřejné proměnné následující argumenty:

- Vstupní (`-i`) a výstupní soubor (`-o`)
- Vstupní soubory s maticemi kGrid (`-k`) a Bell (`-b`)
- Velikost vstupní domény (`-X -Y -Z`)
- Velikost přesahu (`-O`)
- Velikost matic kGrid a Bell (vypočítává)
- Počet opakování hlavní smyčky (`-r`)
- Rozdělení domény – topologii (`-x -y -z`)
- Nápověda (`-h`)
- Chyba zpracování paramerů

Zpracování je implementováno pomocí knihovny Getopt[28]. Kde je očekáván název souboru, argumentem je řetězec, jinak jsou argumenty číselné, nápověda žádnou další hodnotu nevyžaduje.

Zpracování argumentů příkazové řádky se vyvolá voláním konstruktoru třídy `Params` se třemi parametry – prvním je počet argumentů (`argc`), druhým řetězec argumentů a jako třetí se předává řetězec povolených argumentů, který je následně předán rovněž jako třetí parametr funkci `getopt`. Toto řešení bylo zvoleno pro svoji variabilitu, protože každá ze tří variant dekompozice vyžaduje různé argumenty příkazové řádky.

5.3 Implementační rozdíly jednotlivých variant

Vycházejme z toho, že jako základní verzi budeme považovat Fourierovu transformaci využívající lokální dekompozici dat, tedy čistě MPI program bez použití vláken OpenMP. Všechny tři varianty se liší jen a pouze v implementaci třídy `IMatrix`, byly proto vytvořeny tři implementační verze: `IMatrix.cpp`, `IMatrix_OMP.cpp` a `IMatrix_global.cpp`. Jako zásadní rozdíl musíme uvést fakt, že přímo FFT počítá pouze globální varianta dekompozice, vše ostatní jsou pouze aproximace, i když se mohou původnímu výsledku velmi blížit.

5.3.1 Hybridní varianta

Hybridní varianta používající vlákna OpenMP se od té základní liší jen a pouze použitím `#pragma omp parallel for` u všech smyček, u nichž to bylo možné a také tím, že knihovna FFTW je k programu připojena ve verzi pro vlákna OpenMP a při vytváření plánů pro FFT je plánovací funkci předána informace, pro kolik vláken má transformaci provádět.

Pokud jde o rozdíl mezi hybridní variantou takovou, při které jeden MPI proces poběží na jedné procesorové patici spouští 8 vláken a druhou variantou, při níž je jeden MPI proces navázán na celý výpočetní uzel a spouští 16 vláken, tak zde se jedná pouze o rozdíl ve spuštění programu a přidělování procesorových jader pro MPI nebo OpenMP.

[28] GNU.org: Parsing program options using getopt, http://www.gnu.org/software/libc/manual/html_node/Getopt.html

5.3.2 Globální varianta

Globální varianta dekompozice se od té lokální liší především tím, že nepoužívá komunikaci mezi procesy implementovanou ve třídě `PartialMatrix`, tuto třídu využívá pouze pro uložení části vstupních dat. Globální varianta nemá žádné překryvné plochy a díky tomu neprovádí žádný redundantní výpočet. K základní dekompozici na desky přistupujeme tak, že dělíme doménu podle osy Z.

Další rozdíl je opět ve vytváření plánů pro `FFT`, kdy se musí nejdříve vypočítat, jak velká se musí alokovat pole, aby bylo možné provést transpozici matice. Následně se vytvoří plány pro FFT voláním mírně jiné funkce než v minulém případě, stejně tak vykonávání `FFT` a `IFFT()`. Funkční rozdíl byl popsán výše v kapitole 4.

5.4 Testování

Pro testování nově vzniklého řešení Fourierovy transformace využívající lokální dekompozici dat bylo nutné vytvořit spustitelné programy, pomocí kterých by bylo možné ověřit správnost výpočtu, ale také změřit výkonnost a porovnat ji s metodou globální dekompozice. Vznikly proto čtyři spustitelné programy, několik funkcí a skriptů pro Matlab a také několik skriptů pro linuxový shell.

5.4.1 Testovací programy

Pro testování byl zvolen výpočet Laplaciánu pomocí spektrální metody. Jako vzor posloužila implementace v Matlabu, zdrojový kód je k dispozici v příloze 3 tohoto dokumentu. Způsob výpočtu pomocí lokální dekompozice na jedné subdoméně ilustruje výňatek kódu na obrázku 5.5, pro srovnání je na obrázku 5.4 uveden způsob výpočtu globální dekompozice.

Pak byl stejný výpočet implementován v jazyce C++ s využitím lokální Fourierovy dekompozice spektrálních metod. Základní implementace využívající pouze komunikaci pomocí MPI se nachází v souboru `main_localmpi.cpp`, po překladu pomocí `make` vznikne spustitelný soubor `LocalMPI`. Ověření správnosti výpočtu probíhalo tak, že byl nad příslušnými vstupními daty spouštěn program `LocalMPI` s parametrem `-r 1` (pouze jedno opakování smyčky obsahující výpočet Laplaciánu) a výstupy byly porovnány s výsledky stejného kódu v Matlabu.

```
dfdx = real(ifftn( fftshift(1i*kgrid_full.kx).* fftn(f) ));
dfdy = real(ifftn( fftshift(1i*kgrid_full.ky).* fftn(f) ));
dfdz = real(ifftn( fftshift(1i*kgrid_full.kz).* fftn(f) ));
```

Obrázek 5.4 Ilustrace výpočtu Fourierovy transformace využívající globální dekompozici dat.

```
dfdx_sub{x, y, z} =   real(ifftn( ifftshift(1i*kgrid.kx).*
                      fftn(bsxfun(@times, bell_x, f_sub{x, y, z})) ));

dfdy_sub{x, y, z} =   real(ifftn( ifftshift(1i*kgrid.ky).*
                      fftn(bsxfun(@times, bell_y, f_sub{x, y, z})) ));

dfdz_sub{x, y, z} =   real(ifftn( ifftshift(1i*kgrid.kz).*
                      fftn(bsxfun(@times, bell_z, f_sub{x, y, z})) ));
```

Obrázek 5.5 Ilustrace výpočtu Fourierovy transformace využívající lokální dekompozici dat pro jednu subdoménu.

Po ověření správnosti výpočtu základní verze byla implementována hybridní verze výpočtu Laplaciánu využívající paralelní vlákna OpenMP. Implementace této verze se nachází v souboru

main_localhybrid.cpp, po překladu vznikne spustitelný program LocalHybrid. Pro tuto verzi byla rovněž ověřena správnost výpočtu a to stejným způsobem jako v předchozím případě. V obou výše zmíněných variantách je využito překrytí komunikace a výpočtu.

Pro porovnání výkonu lokální a globální dekompozice musela být implementována i samotná globální dekompozice, aby bylo možné testovat stejný výpočet nad stejnými daty, především co se týče náročnosti výpočtu, práce s poli atd. Implementace globální dekompozice se nachází v souboru main_globalmpi.cpp, po překladu vznikne spustitelný program GlobalMPI. Tato verze slouží pouze pro srovnání výkonu a není u ní ověřena korektnost výpočtu.

Pro otestování propustnosti a vůbec doby komunikace při výměně dat mezi procesy byl vytvořen soubor main_benchmark.cpp, který je obdobou základní varianty metody lokální dekompozice, ale v hlavní smyčce neprobíhá vůbec žádný výpočet, pouze výměna dat mezi procesy.

Testování výkonnosti se provádí vícenásobným spuštěním smyčky, ve které je prováděna komunikace a výpočet Fourierovy transformace. Počet opakování je určen argumentem -r. Pro pohodlné měření výkonnosti je možné obejít načítání vstupních souborů a vstupní matice naplnit vygenerovanými daty. To se provede prostým nezadáním vstupních souborů.

5.4.2 Funkce pro Matlab

Aby bylo možné testovat výpočet nad reálnými daty prakticky libovolné velikosti domén a různé topologie dekompozice, je nutné mít připravené matice KGrid a Bell příslušných velikostí, stejně tak i vstupní matice. Aby nebylo nutné je generovat ručně, byly vytvořeny funkce pro Matlab, které generují HDF5 soubory s požadovanými maticemi požadovaných rozměrů. Tyto funkce mají názvy H5SaveBell, H5SaveInput a H5SaveKGrid jsou implementovány v příslušných *.m souborech. Tyto funkce přijímají jako vstupní parametry požadované rozměry matic a případně velikost překryvné oblasti. Výstupem je soubor s požadovanou maticí, cesta k němu je předána jako návratová hodnota.

Funkce jsou implementovány tak, že pokud již existuje soubor, který má obsahovat matici požadované velikosti, matice se negeneruje ani neukládá znovu. Toto chování bylo implementováno proto, aby se při vícenásobném spuštění negenerovala stejná data vícekrát a ušetřil se tak čas.

Všechny výše zmíněné funkce využívají funkce z balíku k-Wave pro Matlab, je tedy nutné, aby byla cesta k adresáři s k-Wave uložena v proměnné Path v programu Matlab.

V budoucích verzích by měly tyto funkce (generování matic KGrid a Bell) součástí zdrojového kódu programu, aby nebylo nutné generovat vstupní matice externě, čímž by se zvýšila variabilita programu.

5.4.3 Skripty pro shell

Pro jednodušší testování vzniklo také několik skriptů pro linuxový shell. Samozřejmostí jsou skripty pro plánovač úloh PBS. Skripty pro PBS nespouštějí přímo konkrétní binární soubory, ale volají je pomocí připravených skriptů. Tím je dosaženo jejich lepší přehlednosti. Ukázky těchto skriptů je možné nalézt ve složce pbs.

Protože spustitelné soubory vyžadují velké množství různých argumentů z příkazové řádky, byly vytvořeny skripty, jejichž název obsahuje klíčové slovo run (například h_run.sh). Tyto skripty přepočítají vstupní parametry, zadají vytvoření vstupních matic správných velikostí (pokud je požadováno) a toto vše předají jako argumenty spustitelnému souboru. Volitelně lze všechny skripty

spouštět i se zapnutým profilerem Allinea[29]. Pokud název skriptu začíná slovem `gen`, skript nevytváří žádná vstupní data a ani nejsou ukládána výstupní data, program pouze vypíše informace o běhu, časy a případná chybová hlášení.

Skript `create_inputs.sh` slouží pro generování reálných vstupních dat do souborů HDF5. Jako parametry jsou mu zadány požadované rozměry domény, topologie mřížky a velikost překryvné oblasti. Podle zadaných parametrů pak skript vytvoří ve složce `indata` a jejích podsložkách příslušné soubory s maticemi. Data jsou vytvářena pomocí funkcí pro Matlab zmíněných v kapitole 5.4.2.

Protože lze z příkazové řádky spouštět pouze skripty pro Matlab, nikoli však přímo volat funkce, byl vytvořen skript `matlab_batch.sh`, který přijímá dva parametry. Prvním je jméno funkce, druhým je řetězec vstupních parametrů, které chceme při volání funkce předat. Skript pak vytvoří dočasný soubor `tmp_matlab_command.m`, který obsahuje pouze volání požadované funkce s příslušnými parametry. Tento soubor už může být spuštěn programem Matlab přímo z příkazové řádky. Po použití je dočasný soubor smazán. V tomto skriptu je dobré nastavit cestu k balíku k-Wave pro Matlab, aby mohly být použity i funkce z k-Wave.

[29] Allinea Performance Reports http://www.allinea.com/products/allinea-performance-reports

6 Experimentální ověření a měření

Důležitou součástí této práce je nejen implementace Fourierovy transformace využívající lokální dekompozici dat, ale i ověření její správnosti a srovnání se současně používanou metodou globální dekompozice. Protože bylo implementováno více variant, bude uvedeno srovnání všech. Před samotným srovnáním výkonu jednotlivých variant budou zmíněny některé faktory, které mají na dobu výpočtu značný vliv.

Pro lepší přehlednost používají všechny grafy porovnávající jednotlivé varianty dekompozice jednotné barevné značení. Červeně a slovem **Global** je značena globální dekompozice, modře a slovem **MPI** je značena lokální dekompozice pouze s využitím MPI, zeleně a slovem **Socket** je značena hybridní lokální dekompozice (používající vlákna OpenMP) taková, kdy každá subdoména připadá na jednu procesorovou patici (8 jader), a žlutou barvou a slovem **Node** je značena hybridní lokální dekompozice, kdy každá subdoména připadá na jeden výpočetní uzel (16 jader).

Pokud nebude přímo uvedeno jinak, všechny výkonnostní testy byly prováděny nad programem implementujícím výpočet Laplaciánu a kvůli úspoře času (tedy i výpočetních prostředků) nebyla používána reálná vstupní data, nýbrž vygenerované hodnoty, stejně tak nebyly ukládány žádné výstupní matice. Při všech měřeních uvedených v této práci byly funkce pro vytváření plánů volány s příznakem `FFTW_MEASURE`.

6.1 Ověření korektnosti výpočtu

Aby byla nová metoda v praxi použitelná, musí poskytovat korektní výsledky. U této metody přesnost úzce souvisí s velikostí jednotlivých subdomén a velikostí překryvné oblasti. Obecně platí, že čím větší, tím přesnější výpočet. Grafy podkládající toto tvrzení jsou uvedeny už v kapitole 4.3.1. Vliv na přesnost výpočtu má i matice `Bell`, jejíž co nejpřesnější výpočet je stále předmětem výzkumu tvůrců programu k-Wave.

Při implementaci byla korektnost výsledků kontrolována porovnáním výstup programu po jedné iteraci výpočtu s referenčním řešením v Matlabu a byla sledována maximální relativní odchylka[30] obou řešení. Při testování na menších doménách (rozměr 256^3) byla jako přípustná stanovena hodnota odchylky 10^{-4} a menší. Pokud bychom chtěli dosáhnout větší přesnosti, mohli bychom zvětšit velikost domény či překryvné oblasti.

6.2 Faktory ovlivňující výkon

V průběhu měření bylo zjištěno, že doba výpočtu nenarůstá přímo úměrně k rostoucí velikosti domény, stejně jako neklesá přímo úměrně k počtu výpočetních jader. Některé faktory způsobující tento jev byly nalezeny a jejich vliv změřen, některé měřit nelze (například vliv zatížení stroje systémovými úlohami).

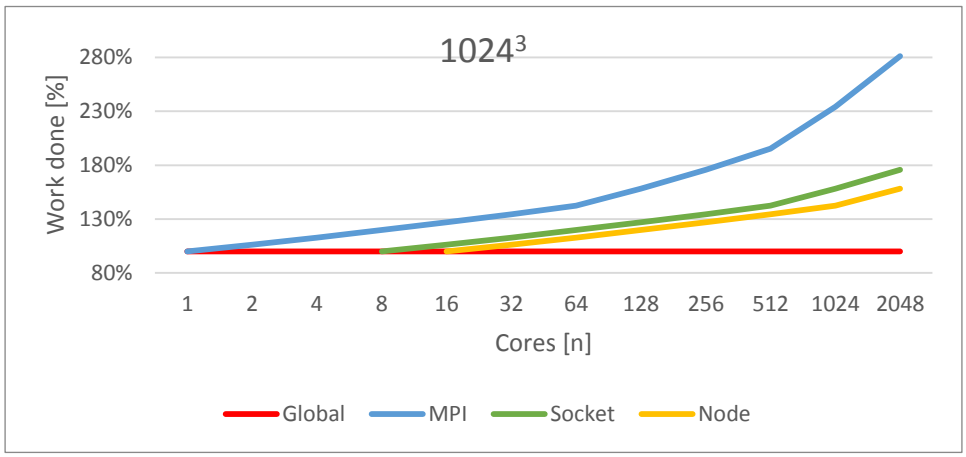
6.2.1 Redundance výpočtu

V kapitole 4.3.2 je uveden graf redundance výpočtu v závislosti na velikosti překryvu, na tento problém se ale můžeme podívat i z druhé strany, tedy jaká je redundance výpočtu při měnícím se

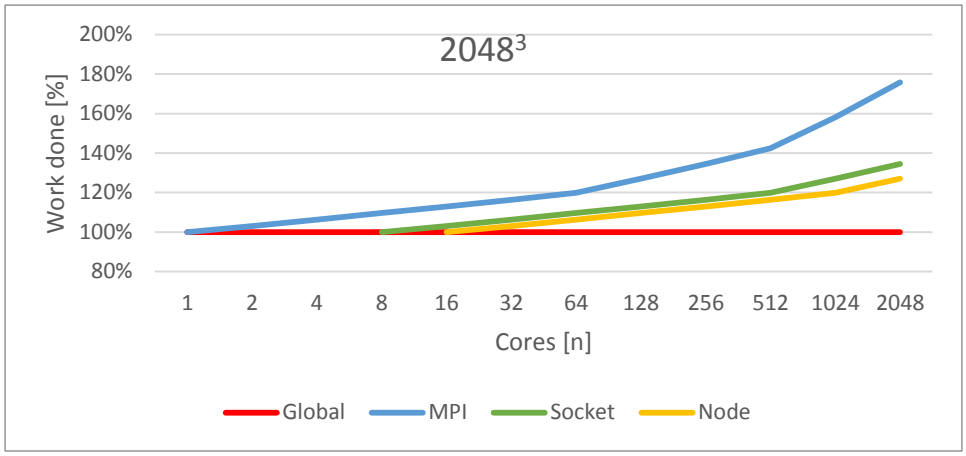
[30] $L_{infinity} = maximální\ relativní\ odchylka = \frac{maximální\ hodnota\ odchylky}{maximální\ hodnota\ v\ referenčním\ řešení}$

počtu výpočetních jader a konstantní velikosti překryvu. Zohledněn je i fakt, že při hybridní dekompozici jedna subdoména připadá buď na procesorovou patici, nebo výpočetní uzel.

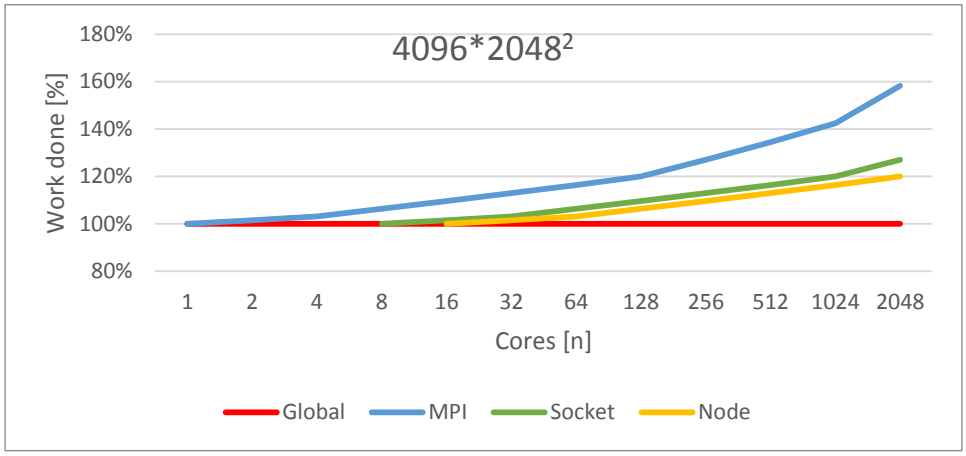
Pro následující hodnoty platí, že nižší je lepší.



Obrázek 6.1 Redundance výpočtu při různých dekompozicích domény o velikosti 1024*1024*1024.



Obrázek 6.2 Redundance výpočtu při různých dekompozicích domény o velikosti 2048*2048*2048.



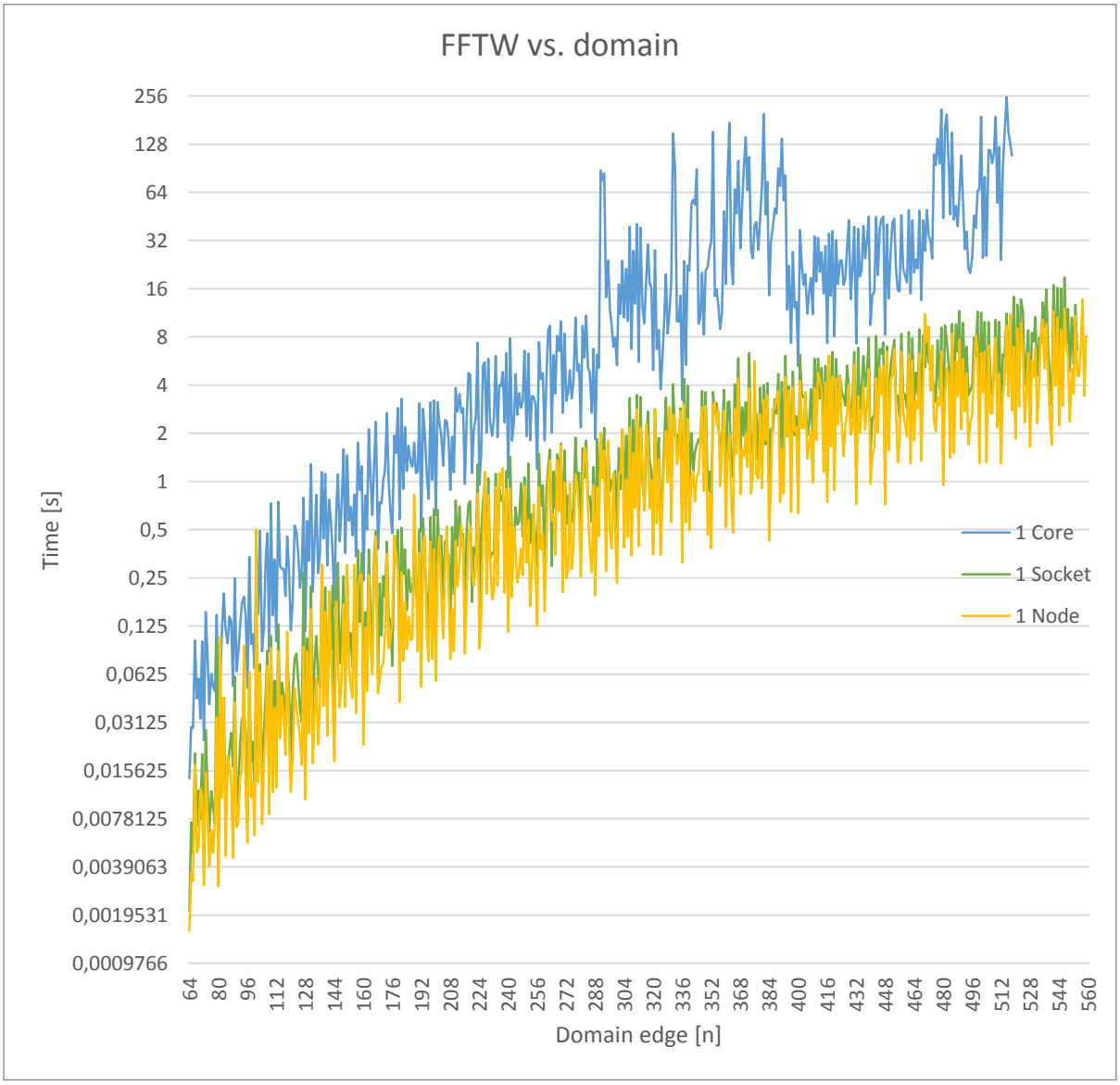
Obrázek 6.3 Redundance výpočtu při různých dekompozicích domény o velikosti 2048*2048*4096.

6.2.2 Optimalizace knihovny FFTW

Jak uvádí dokumentace knihovny FFTW, nejrychleji se počítá Fourierova transformace nad doménami o velikosti násobku mocnin malých prvočísel [21]. Bylo tedy provedeno měření, jehož účelem bylo zjistit, jak dlouho trvá jednomu jádru, patici nebo uzlu vypočítat FFT nad doménami různých velikostí. Použit byl opět výpočet Laplaciánu, neprobíhaly žádné MPI komunikace. Výsledky prezentuje graf na obrázku 6.4. Tabulka zdrojových dat k tomuto grafu je uvedena jako příloha číslo 4 tohoto dokumentu.

Z grafu je jasně patrné, že doba výpočtu FFT knihovnou FFTW pro 3D doménu opravdu velmi kolísá, v extrémních případech je čas výpočtu pro doménu o velikosti $(n+1)^3$ klidně až 15krát delší, než pro doménu o velikosti n^3, kdy absolutně nejhorších hodnot je dosahováno pro velká prvočísla.

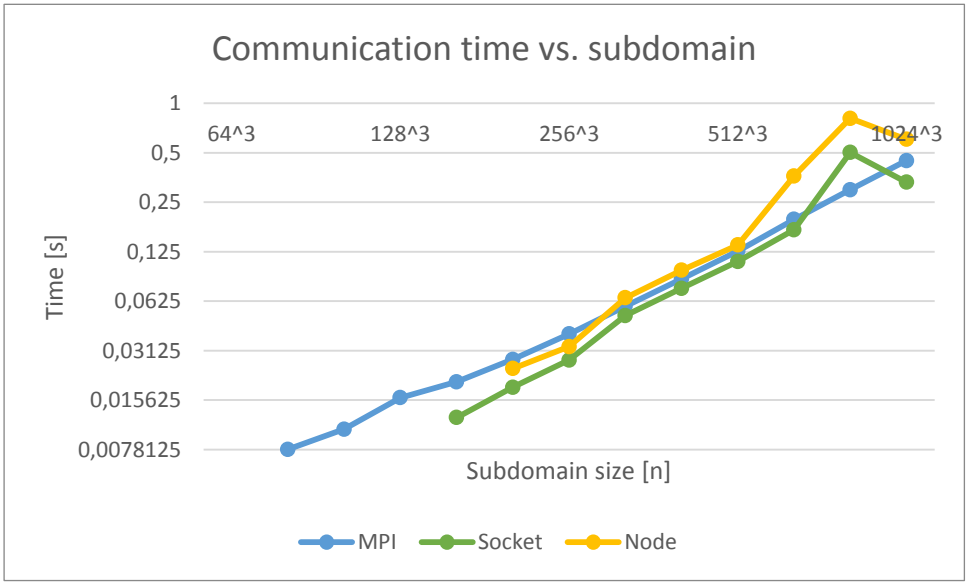
Vzhledem k faktu, že knihovna FFTW vybírá nejvhodnější algoritmus výpočtu znovu při každém spuštění, mohou se dosažené výsledky lišit i při několikanásobném spuštění stejné úlohy, výsledky těchto měření je tak nutné brát v potaz s jistou rezervou.



Obrázek 6.4 Doba výpočtu FFT na různých velikostech domény. Časová osa (svislá) je logaritmická.

6.2.3 Rychlost MPI komunikace vůči velikosti subdomény

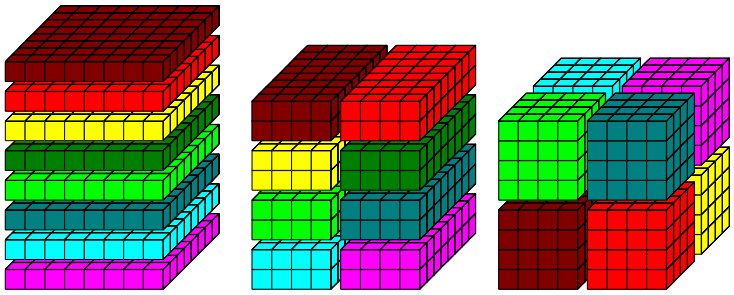
Cílem tohoto měření bylo zjistit, jak roste doba MPI komunikace s rostoucí velikostí subdomény. Pro testování byla zvolena nejmenší možná použitelná topologie, tedy 2*2*2, každý proces dostal stejně velkou subdoménu (plus překryv o konstantní velikosti 16). Testovaná subdoména (bez překryvu) se pro každý krok měření zvětšila dvojnásobně, se zvětšující se subdoménou klesá poměr velikosti subdomény včetně ohraničení vůči samotné subdoméně (režie výpočtu). Pro toto měření byl použit program `LocalMPIBenchmark`. Výsledky prezentuje obrázek 6.5.



Obrázek 6.5 Doba výměny dat mezi subdoménami různé velikosti, časová osa (svislá) je logaritmická.

6.2.4 Vliv tvaru subdomény na rychlost výpočtu

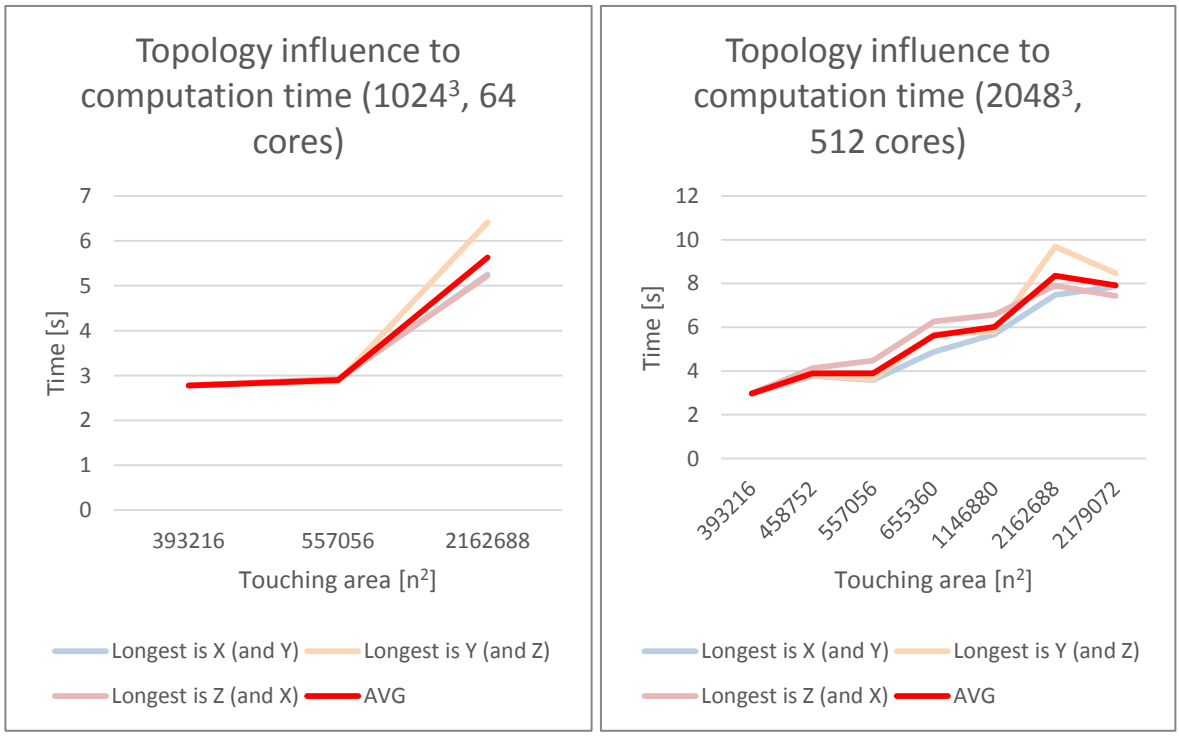
Toto měření mělo potvrdit či vyvrátit domněnku, že ideální tvar subdomény připadající na jeden výpočetní proces by měl být ideálně krychle, protože v tom případě je mezi jednotlivými subdoménami nejmenší styčná plocha a tedy je i nejmenší objem přenášených dat pří výměně překryvných oblastí a nejmenší redundance výpočtu FFT. Základní možná rozdělení domény ilustruje obrázek 6.6



6.6 Lokální 1D (z), 2D (x, z) a 3D (x, y, z) dekompozice domény o velikosti 8*8*8 na 8 subdomén.

Měření byla provedena dvě, jedno nad doménou o velikosti 1024^3 rozdělenou mezi 64 výpočetních jader třemi různými způsoby, druhé nad doménou 2048^3 rozdělenou mezi 512 výpočetních jader sedmi různými způsoby. Objem subdomény (bez překryvu) v rámci měření je vždy stejný, mění se styčná plocha s okolím.

Grafy na obrázcích 6.7 a 6.8 zobrazují výsledky tohoto měření a potvrzují původní domněnku, že ideálním tvarem subdomény je tvar co nejbližší krychli. Jak se zvětšuje styčná plocha s okolními subdoménami, narůstá i doba výpočtu. V grafech je na vodorovné ose vynesen povrch kvádru (bez překryvu), jež subdoména tvoří, nejnižší hodnoty se tedy nejvíce blíží krychli, nejvyšší odpovídají tenké desce. Měření probíhala pro každý rozměr třikrát, protože může záležet i na otočení kvádru v prostoru. Zobrazeny jsou všechny tři výsledky i průměrná hodnota (AVG).

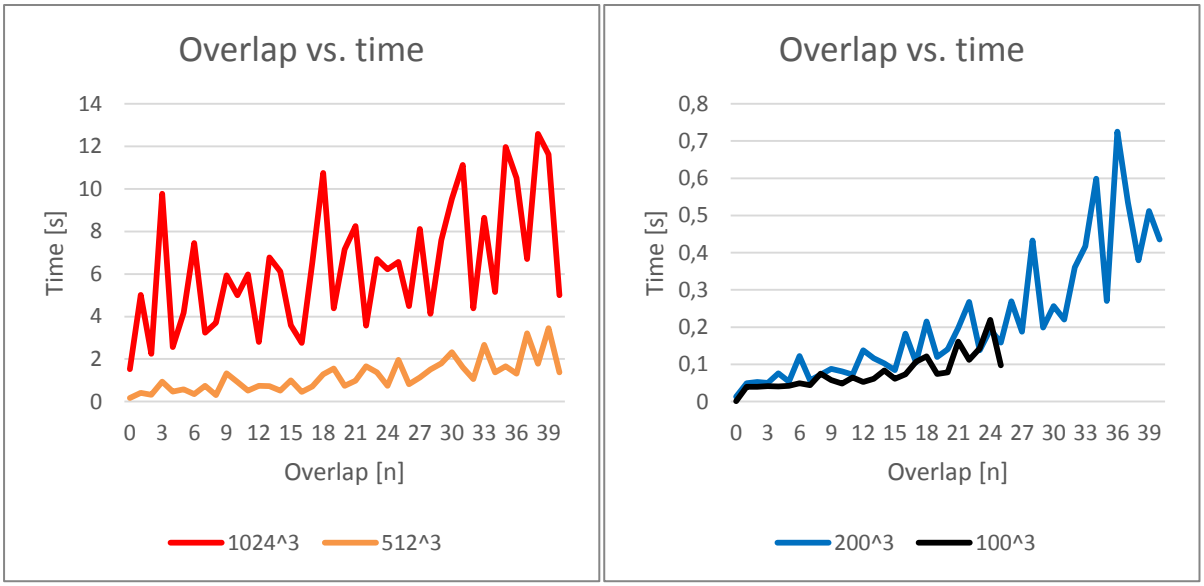


Obrázek 6.7 Vliv tvaru subdomény na dobu výpočtu – doména o velikosti 1024^3, 64 výpočetních jader.

Obrázek 6.8 Vliv tvaru subdomény na dobu výpočtu – doména o velikosti 2048^3, 512 výpočetních jader.

6.2.5 Vliv velikosti překryvu na dobu výpočtu

V kapitole 4.3.1 byl uveden graf vlivu velikosti překryvné oblasti na přesnost výpočtu. Nyní se již můžeme zabývat i druhou stránkou tohoto problému, tedy vlivem velikosti překryvné oblasti na dobu výpočtu. Na výsledcích se sice značně projevuje vliv optimalizace knihovny FFTW, ale z grafů 6.9 a 6.10 lze vyčíst jistý stoupající trend.



Obrázek 6.9 Vliv velikosti překryvu na dobu výpočtu, 64 jader, větší velikosti domény.

Obrázek 6.10 Vliv velikosti překryvu na dobu výpočtu, 64 jader, menší velikosti domény.

6.3 Srovnání výkonnosti

Po vysvětlení si některých faktorů ovlivňujících dobu výpočtu se konečně dostáváme k samotnému srovnání, kterému tak můžeme lépe porozumět.

6.3.1 Weak scaling

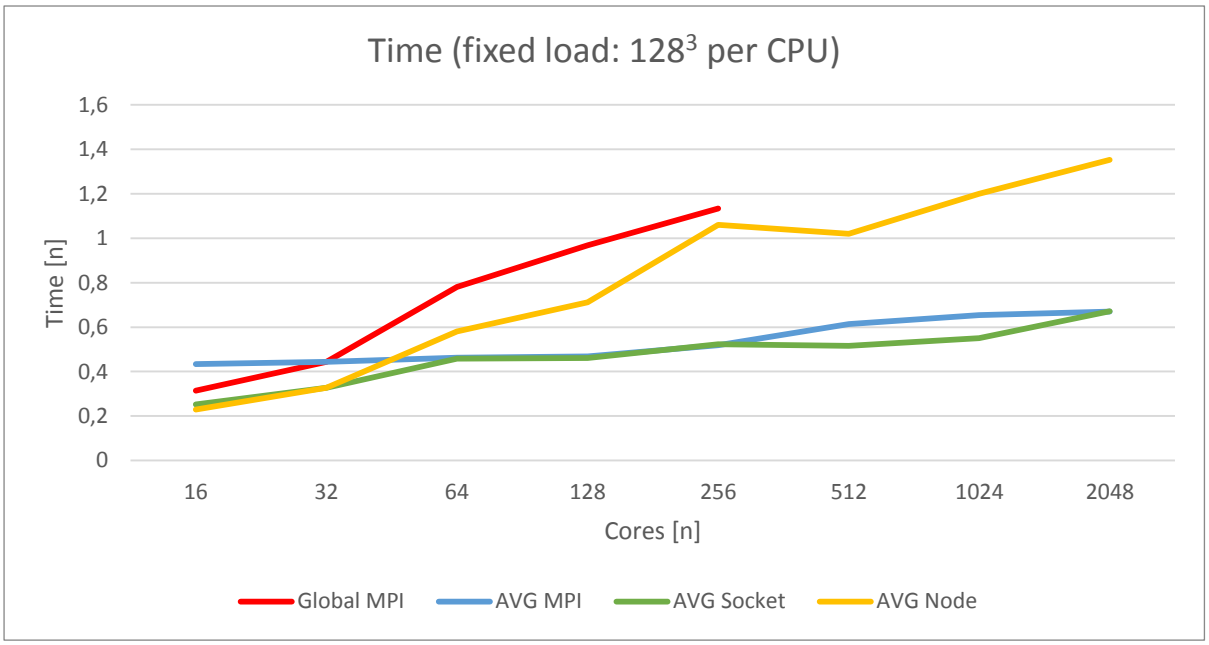
Prvním ze srovnání, která budou na následujících stranách prezentována, je tzv. slabé škálování. Sledovaným parametrem je zde již tradičně čas, měřena byla doba výpočtu problému na 2^n výpočetních jádrech, přičemž velikost problému roste lineárně s počtem výpočetních jader (velikost podproblému připadajícího na jedno jádro je tedy fixní).

Pro toto měření byla jako základní velikost problému připadající na jedno jádro zvolena hodnota 128^3 a velikost překryvu (overlap) 16. Měření bylo prováděno na 1 až 128 výpočetních uzlech (tedy 16 až 2048 jádrech), počet jader byl vždy mocnina čísla 2. Topologie dekompozice byla vždy taková, aby tvar subdomény byl co nejbližší ideálnímu tvaru – krychli (viz kapitola 6.2.4), čehož bohužel nelze vždy dosáhnout jen jedním způsobem, často tedy bylo prováděno více měření, kdy byla subdoména stejné velikosti vždy otočena podle jiné osy. Výsledky těchto více měření byly ve výsledku pro každou velikost zprůměrovány.

Bohužel se nepodařilo změřit dobu výpočtu metody globální transpozice pro více než 512 výpočetních jader, protože při vyšších počtech se program zasekl na volání funkce z knihovny FFTW pro výpočet velikosti polí, jež je nutno alokovat.

Srovnání délky výpočtu jedné iterace ukazuje graf na obrázku 6.11. Je-li velikost problému přepočtená na jedno jádro fixní, dalo by se očekávat, že i doba výpočtu bude fixní, což splňuje MPI varianta lokální dekompozice a hybridní varianta lokální dekompozice, při níž je subdoména navázána na patici (socket). Lepší hodnoty při nízkém počtu jader u obou hybridních variant jsou způsobeny tím, že MPI komunikace do všech směrů zde probíhá až od počtu 64, respektive 128 jader. Špatné výsledky druhé hybridní varianty (Node) můžeme přisoudit nerovnoměrnému přístupu

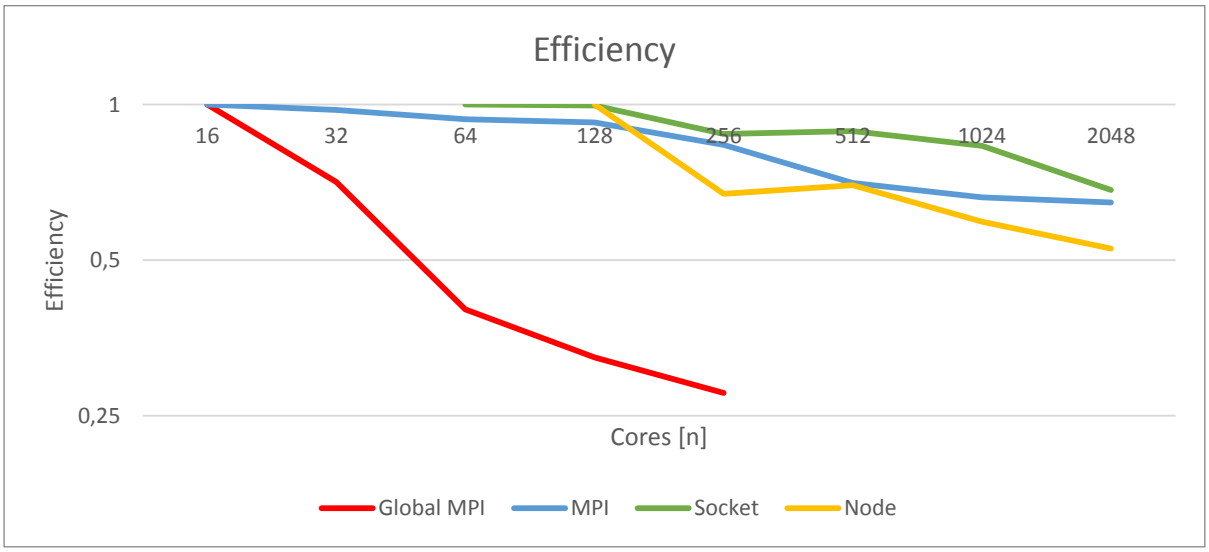
do operační paměti nebo také optimalizaci FFTW. Horší výsledek globální varianty byl očekáván, protože zde dochází ke většímu množství komunikace – musí se přeposlat celá doména tam i zpět.



Obrázek 6.11 Weak scaling – srovnání doby výpočtu.

Kromě doby výpočtu může být zajímavý i graf efektivity výpočtu. Výchozí hodnotou pro každé srovnání v tomto grafu je vždy takový nejmenší počet jader, na kterých lze provést dekompozici s takovou topologií, aby probíhala MPI komunikace všemi možnými směry (topologie $[2, 2, 2]$), případně nejmenší změřený počet jader.

Jak ukazuje graf na obrázku 6.12 efektivita výpočtu lokální MPI verze a lokální hybridní verze (Socket) je i se stoupajícím počtem výpočetních jader velmi dobrá, i když se samozřejmě postupně začíná projevovat nutná synchronizace většího množství procesů a případné zahlcení linek.



Obrázek 6.12 Weak scaling – efektivita (svislá osa je logaritmická).

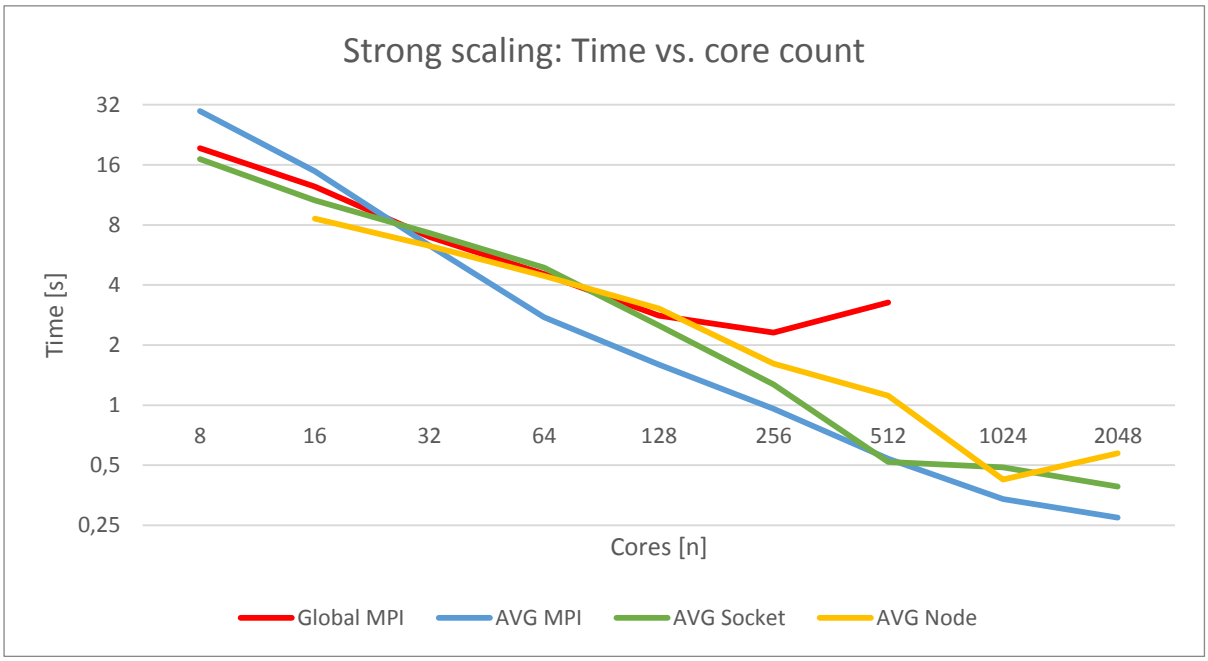
6.3.2 Strong scaling

Takzvané silné škálování sleduje stejně jako předchozí metrika dobu výpočtu, nyní je však fixní celková velikost problému a přidáváním jader se snažíme docílit zkrácení doby výpočtu. V ideálním případě by zrychlení rostlo lineárně s počtem výpočetních jader, toho však vlivem nutné synchronizace, režie a redundance výpočtu nelze žádným normálním způsobem dosáhnout.

Měření probíhalo na 1 až 128 výpočetních uzlech, topologie dekompozice byla vždy volena tak, aby každá subdoména měla tvar co nejvíce blížící se krychli (viz kapitola 6.2.4). Měření globální metody na více než 512 výpočetních jádrech se nepodařilo realizovat, neboť se program při vyšších hodnotách zasekl na volání funkce z knihovny FFTW pro výpočet velikosti polí, jež je nutno alokovat.

6.3.2.1 Velikost problému: 1024^3

Jako první byla zvolena základní velikost problému 1024^3, měření probíhalo na 1 až 128 výpočetních uzlech.

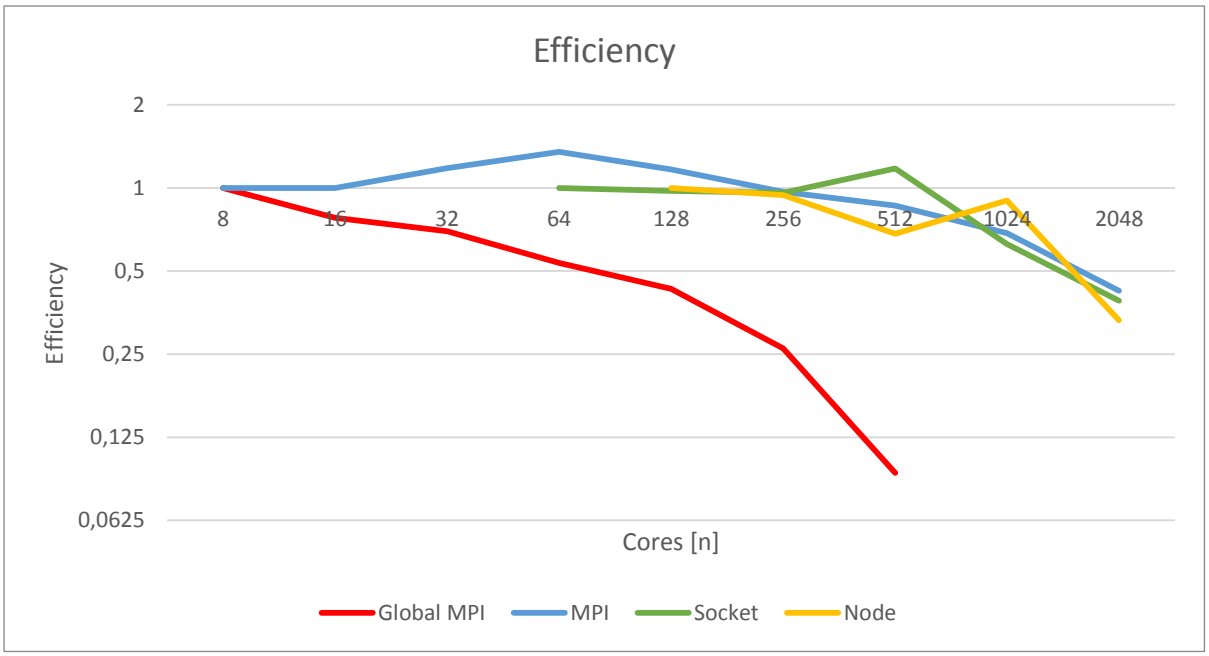


Obrázek 6.13 Strong scaling – doména 1024^3, srovnání doby výpočtu.

Při pohledu na graf doby výpočtu na obrázku 6.13 vidíme, že lokální MPI varianta dekompozice se jeví jako nejlepší z testovaných. Vysvětlit to můžeme opět buď optimalizací FFTW nebo vlivem MPI komunikace, i když je zde objem posílaných dat stále docela malý. V lokální MPI verzi běží 8 procesů na jednom procesoru, tedy zaslání dat až 7 sousedům nemusí opustit procesor, kdežto v hybridních variantách mezi sebou komunikují přímo jednotlivé procesory či uzly, mimo to v přepočtu na MPI proces posílají značně větší objemy dat (dvakrát nebo čtyřikrát delší hrany, čtyřikrát nebo osmkrát větší stěny). Fakt, že globální varianta bude nejhorší, byl očekáván, ale že při určitém počtu jader výrazně zpomalí je překvapivé, ale i vysvětlitelné použitím plánovacího příznaku `FFTW_MEASURE`.

Na grafu efektivity na obrázku 6.14 je zajímavé, že efektivita lokální MPI verze a hybridní (socket) verze v některých bodech překračuje hodnotu 100 %. To může být způsobeno opět optimalizací FFTW, protože jako základní hodnota je brána dekompozice na pouhých 8 subdomén,

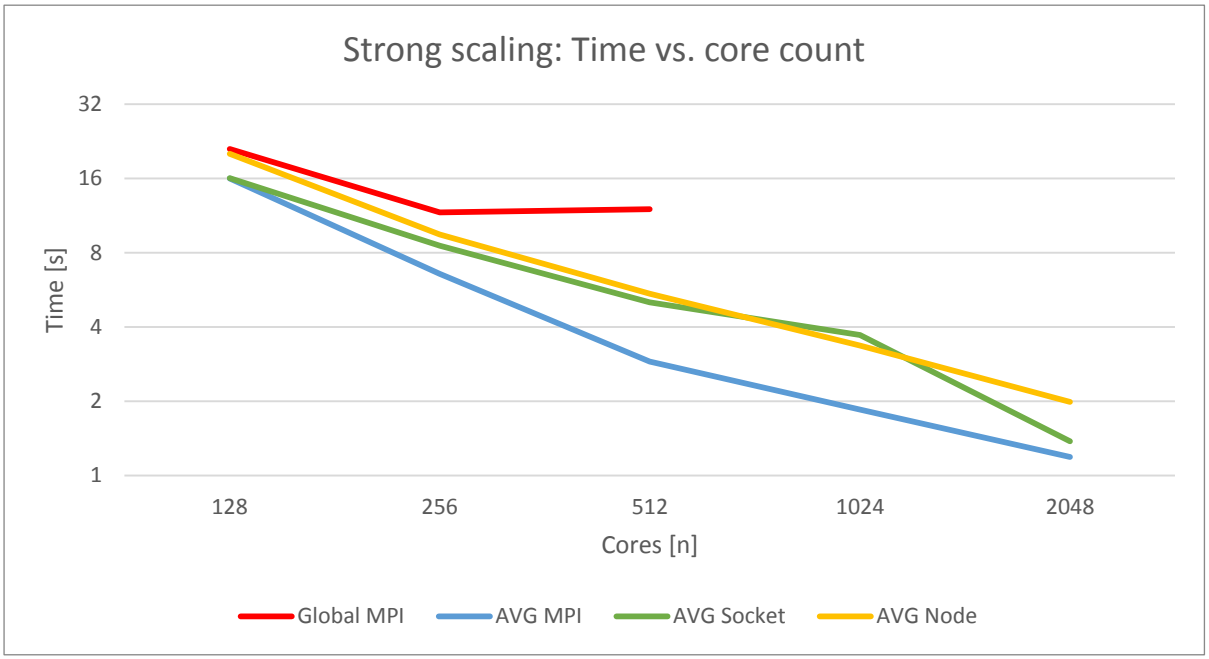
které jsou značně velké, a doba výpočtu FFT je tedy také delší. Vliv má i tvar subdomény, když v těch případech, kdy efektivita roste, má přesný tvar krychle.



Obrázek 6.14 Strong scaling – doména 1024^3 – efektivita výpočtu.

6.3.2.2 Velikost problému: 2048^3

Jelikož se jedná o osminásobně větší problém než v předchozím případě, nemá smysl měřit škálování na příliš malém počtu jader, bylo tedy měřeno nejméně na 8 uzlech.

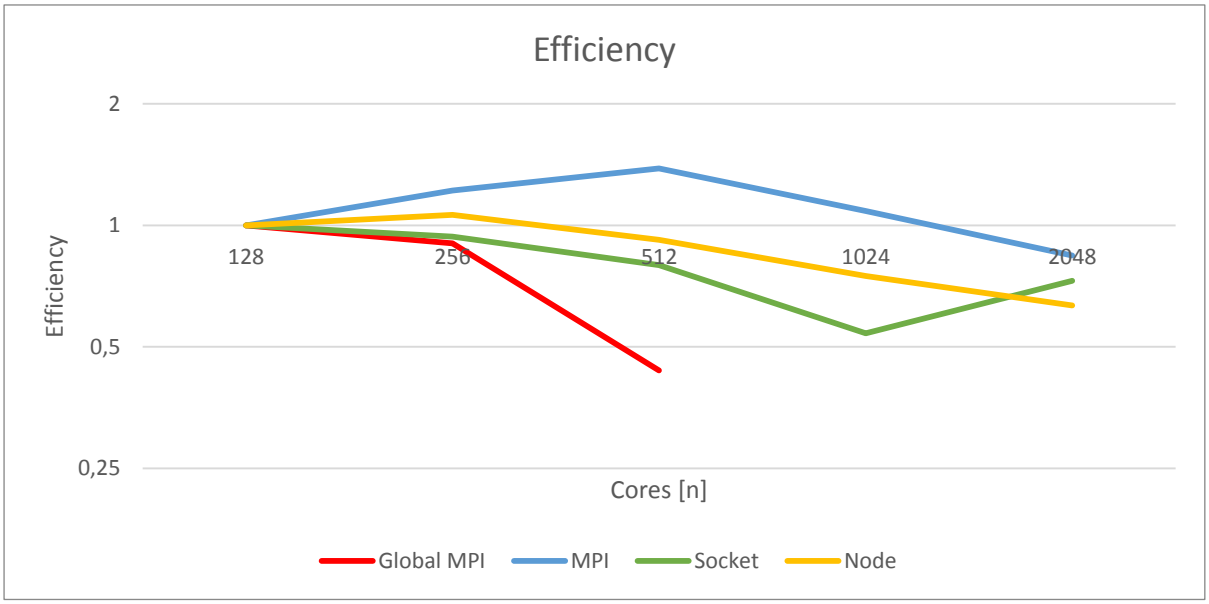


Obrázek 6.15 Strong scaling – doména 2048^3, srovnání doby výpočtu.

Po vyhodnocení předchozího měření byly očekávány i zde podobné výsledky, což, jak je patrné z grafu na obrázku 6.15, se i splnilo. Dokonce, vezmeme-li grafy a položíme je přes sebe, můžeme

vidět jistou podobnost v místech, v nichž na každý proces připadá stejně velká subdoména (například čas pro 512 jader v tomto měření a čas pro 64 jader v předchozím). Z toho lze vyvodit, že počet komunikujících procesů nemá tak velký vliv na dobu výpočtu jako velikost subdomény připadající na jedno výpočetní jádro.

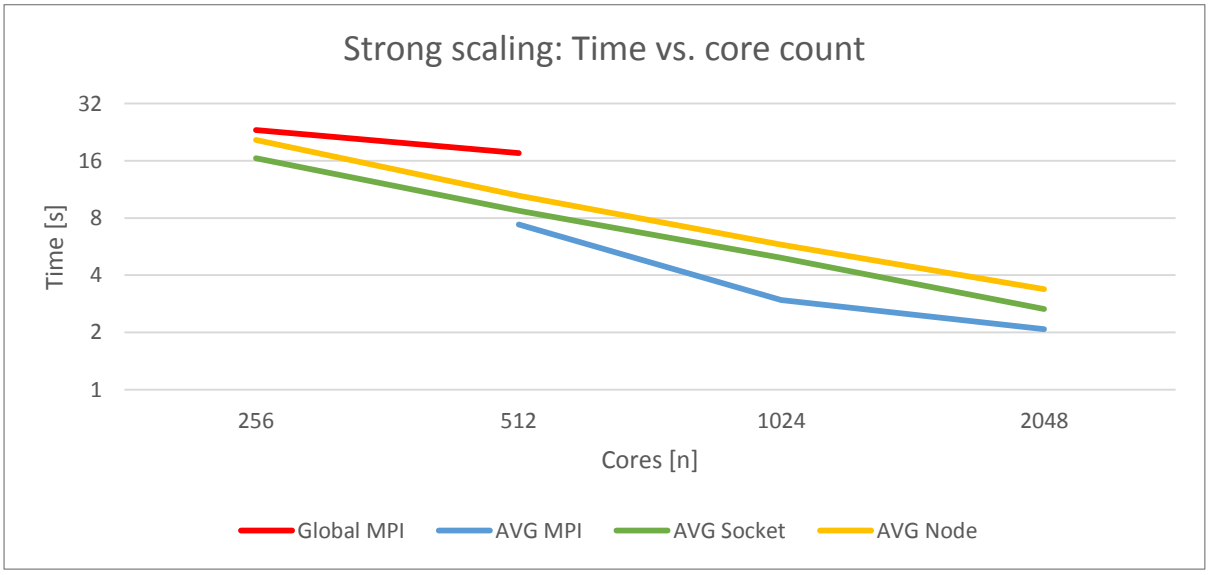
Graf efektivity výpočtu (obrázek 6.16) je opět silně poznamenán optimalizací knihovny FFTW a vlivem tvaru domény na dobu výpočtu, jinak nelze vysvětlit efektivitu vyšší než 100 %.



Obrázek 6.16 Strong scaling – doména 2048^3 – efektivita výpočtu.

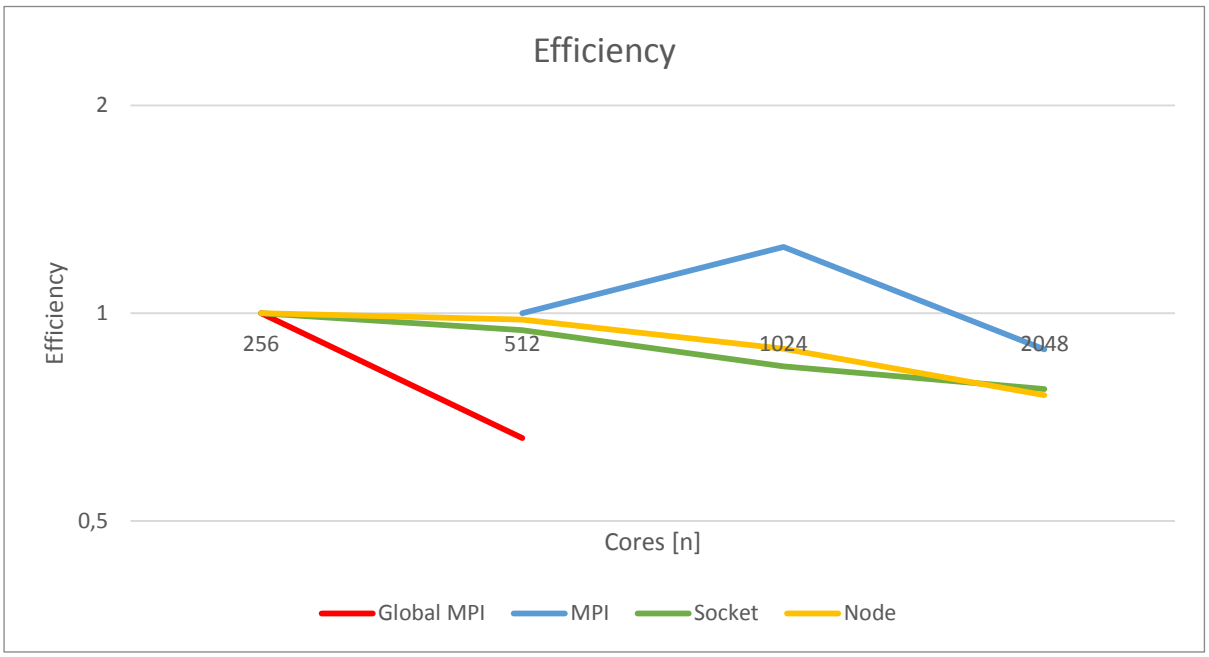
6.3.2.3 Velikost problému: 4096*2048^2

Pro poslední měření silného škálování byla velikost problému ještě dvakrát zvětšena. V tomto případě se bohužel nepodařilo změřit lokální MPI dekompozici pro 256 jader z důvodu nedostatku operační paměti na uzlu.



Obrázek 6.17 Strong scaling – doména 4096*2048^2, srovnání doby výpočtu.

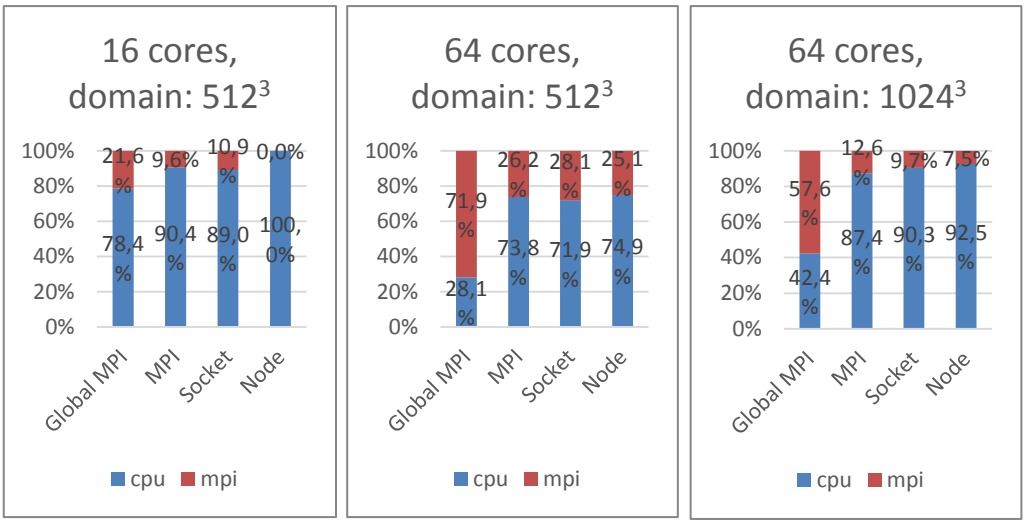
I v grafu na obrázku 6.17 lze sledovat podobný vývoj jako v předchozím měření, pro stejnou velikost subdomény si zhruba odpovídají i doby výpočtu. Pohledem na graf efektivity (obrázek 6.18) se můžeme utvrdit v domněnce, že subdoména ve tvaru krychle je pro výpočet nejlepší.



Obrázek 6.18 Strong scaling – doména 4096*2048^2 – efektivita výpočtu.

6.3.3 Profilování

Pro analýzu výpočtu z hlediska výkonnosti může být dobré zjistit, jaký podíl na celkové době výpočtu tvoří MPI komunikace a jaký podíl má samotný výpočet. Proto byly všechny čtyři varianty spuštěny s profilerem Allinea Performance Reports. Při každém spuštění bylo vždy vykonáno alespoň 50 iterací hlavní smyčky, aby bylo upozaděno generování dat a další režie.

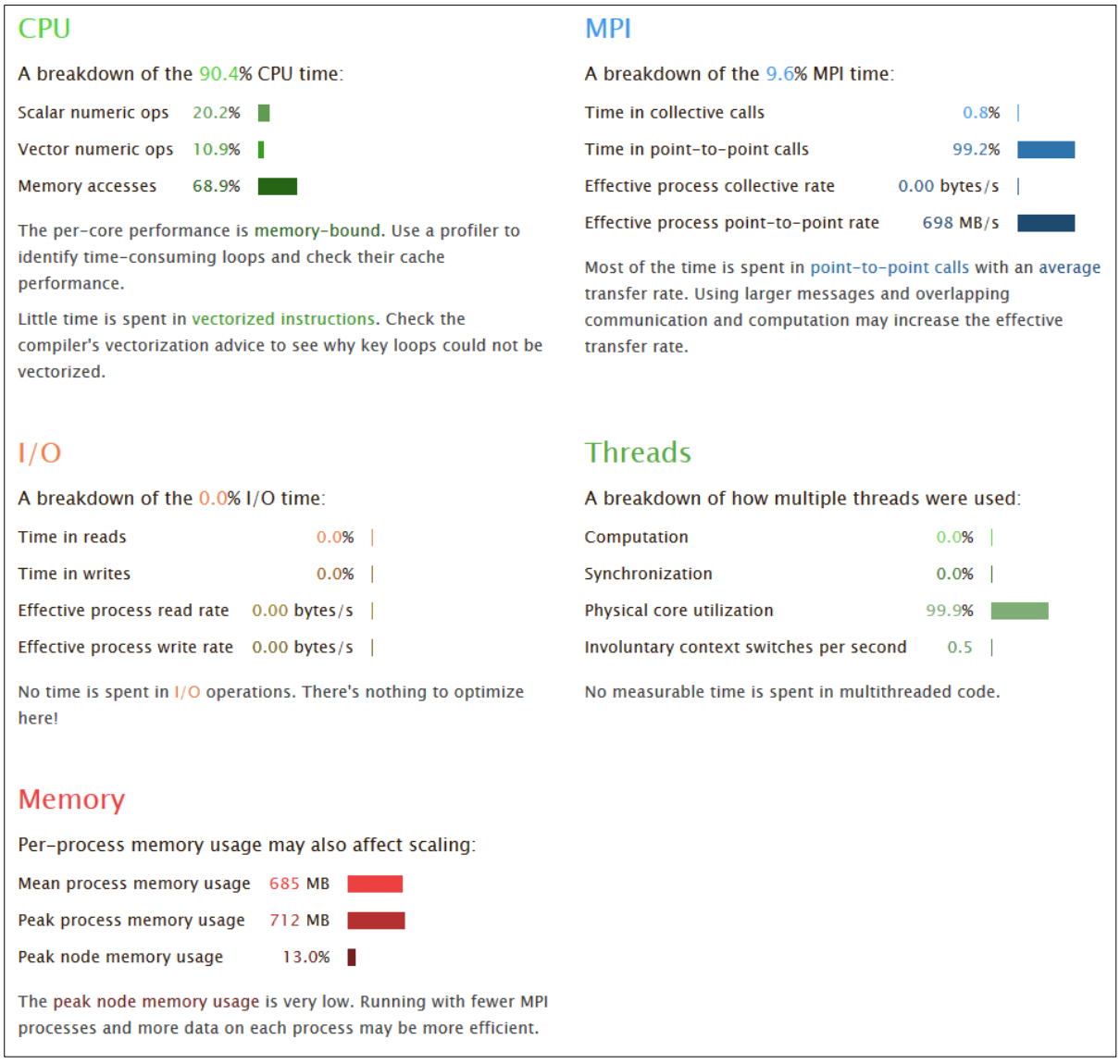


Obrázek 6.19 Výstup profileru Allinea, srovnání jednotlivých variant.

Bohužel použití profileru Allinea je omezeno maximálním počtem 64 MPI procesů a ani u tohoto čísla není jistota, že se jej podaří vždy spustit (mohou jej využívat jiní uživatelé) a jeho chování bude

korektní. Pro výsledky s vyšší výpovědní hodnotou by bylo vhodné použít nějaký jiný profiler (například IPM[31], Vampir[32]), který by uměl měřit například pouze části kódu a nikoli pouze celý běh programu jako celek.

Výstup profileru lze porovnat pro různé velikosti domény či počty jader v grafech na obrázku 6.19. Z něj je patrné především to, že čím větší je subdoména, tím je menší procento času strávené komunikací, což bylo očekáváno. Nulová hodnota v pravém sloupci prvního grafu je způsobena faktem, že 1 uzel má 16 výpočetních jader a žádná MPI komunikace neprobíhá, jelikož je spuštěn pouze jeden MPI proces. Podrobnější výstupy profilování (ukázka na obrázku 6.20) se po spuštění se zapnutým profilováním nachází ve složce `reports`.



Obrázek 6.20 Výstup profileru Allinea Performance Reports při spuštění skriptem `gen_run.sh 512 512 512 2 2 4 16 50 prof`

[31] IPM – Integrated Performance Monitoring, ipm-hpc.sourceforge.net
[32] Vampir, www.vampir.eu

7 Zhodnocení výsledků

Na základě výsledků měření a dalších experimentů uvedených v kapitole 6 si můžeme konečně odpovědět na otázku, zda má metoda Fourierovy transformace využívající lokální dekompozici dat vůbec smysl. Nyní, můžeme říci, že ano, smysl má, ale má i určitá omezení, která je nutné brát při každé konkrétní aplikaci v úvahu. Přednosti metody a některá doporučení budou demonstrována na jednoduchém příkladu.

7.1 Proč používat lokální dekompozici?

Vrátíme-li se ke kapitole 4.1 pojednávající o motivaci pro vznik tohoto nového řešení, můžeme říct, že se podařilo splnit všechny body, které jsou ve zmíněné kapitole vyjmenovány. Ve srovnání s metodou globální dekompozice je lokální dekompozice lepší ve všech bodech.

- **Rychlost výpočtu** – Jak ukazují měření uvedená v kapitole 6.3, Fourierova transformace využívající lokální dekompozici dat v libovolné ze tří možných variant je pro větší domény a větší počty výpočetních jader vždy rychlejší než Fourierova transformace využívající globální dekompozici dat.
- **Variabilita množství výpočetních prostředků** – Jednou ze zásadních nevýhod globální dekompozice je fakt, že počet výpočetních jader nemůže být větší, než nejdelší rozměr domény. Druhým problémem je, že pokud není počet výpočetních jader dělitelem velikosti nejdelšího rozměru domény, nepodaří se všechna jádra ani rovnoměrně vytížit. Oba tyto problémy lokální dekompozice dokáže řešit, i když ten druhý pouze do jisté míry – prvočísla opravdu dělit nelze. Pro lokální dekompozici stačí, když najdeme čísla x, y a z taková, že počet jader/procesorů/uzlů (počet subdomén) $P = x * y * z$ a zároveň jsou tato čísla děliteli jednotlivých rozměrů domény.
- **Zvětšování domény i počtu procesorů** – Dalším problémem globální dekompozice je operační paměť a komunikace. Fakt, že větší doména potřebuje více operační paměti, platí samozřejmě i při lokální dekompozici, samotný problém tkví v transpozici domény, při níž se celá doména musí mezi jednotlivými procesory přeposlat, a jak roste počet procesů, roste i objem komunikace a velikost bufferů pro tuto komunikaci, neboť procesy komunikují každý s každým. Při lokální dekompozici každý proces komunikuje pouze se 26 sousedními procesy, zvětšíme-li tedy doménu a zároveň stejně zvětšíme i množství výpočetních prostředků, paměťová i komunikační náročnost jednoho procesu zůstává stejná.
- **Paměťové a komunikační nároky** – Jak bylo uvedeno v předchozím bodě, při zvětšení počtu výpočetních prostředků nenarůstá paměťová ani komunikační náročnost přepočtená na jeden proces. Může tedy být zajímavé použít toto řešení na výpočetních clusterech složených z mobilních čipů s menším množstvím operační paměti.

7.1.1 Příklad

Na závěr uveďme příklad. Mějme doménu o velikosti $3000 * 3000 * 3000$ bodů a přístup 500 uzlům superpočítače (1 uzel $= 2 * 8$ jader), tedy máme k dispozici celkem 8000 jader. Pokud bychom zvolili globální dekompozici, můžeme použít maximálně 3000 jader. Dále by mohl nastat problém s dostatkem operační paměti a zahlcením komunikační sítě. Při lokální dekompozici můžeme zvolit

například topologii dekompozice $[10, 20, 40]$ nebo $[20, 20, 20]$ a využít všechny výpočetní prostředky.

7.2 Proč nepoužívat lokální dekompozici?

Zásadní problém Fourierovy transformace využívající lokální dekompozici dat je pouze jeden a tím je přesnost výpočtu. Tento problém není ani tak chybou, jako spíše vlastností tohoto řešení, protože se jedná o zcela jiný přístup k výpočtu Fourierovy transformace nad maticí.

Bereme-li jako referenční řešení výpočet FFT nad maticí jedním procesem bez jakékoli dekompozice, pak metoda globální dekompozice vyprodukuje téměř stejný výsledek jako toto referenční řešení, protože provádí úplně stejný výpočet nad stejnými daty jen s tím rozdílem, že je tento výpočet prováděn paralelně. Metoda lokální dekompozice k problému přistupuje jiným způsobem, každý proces provádí výpočet pouze nad částí dat, a i když se tyto části částečně překrývají, nelze zaručit, že výsledek bude totožný. Tato vlastnost vyplývá z podstaty výpočtu Fourierovy transformace.

Nepřesnost výpočtu lze značně omezit zvětšením překryvné oblasti nebo snížením počtu (zvětšením velikosti) subdomén. Pokud je požadován stoprocentně přesný výsledek, je lepší použít jiný způsob výpočtu než lokální dekompozici.

7.3 Jak používat lokální dekompozici?

Z měření provedených v kapitole 6 lze vysledovat jisté vlastnosti Fourierovy transformace využívající lokální dekompozici dat. Budeme-li tyto vlastnosti respektovat, může náš výpočet být velmi přesný a zároveň i velmi rychlý.

Rozhodne-li se uživatel použít Fourierovu transformaci využívající lokální dekompozici dat pro svůj výpočet, měl by se řídit následujícími doporučeními:

- Ideálním tvarem subdomény je krychle. Pokud ideálního tvaru nelze dosáhnout, měl by se tvar krychli alespoň co nejvíce blížit.
- Je nutné brát v úvahu lepší optimalizaci FFTW pro některé rozměry domény. Pokud existuje více možných způsobů dekompozice, pak je dobré nejprve porovnat dobu výpočtu FFT pro každý z nich a podle toho se rozhodnout. Lze například nahlédnout do tabulky, která je přílohou číslo 4 tohoto dokumentu. V tabulce jsou zaznamenány doby výpočtu Laplaciánu (3*FFT + 3*IFFT) nad doménami o rozměru n^3pro všechna celá $n \in \langle 64; 520 \rangle$. Nevhodný rozměr subdomény lze snadno upravit zvětšením překryvné oblasti.
- Požadujeme-li větší přesnost výpočtu, nabízí se jako první možnost zvětšení překryvné oblasti. Nedoporučuje se však překryv zvětšovat pouze o 1, nýbrž o více, a to ze dvou důvodů: (i) Zvětšení o 1 nebude mít nijak významný efekt, (ii) pokud byl dosavadní výpočet rychlý, pravděpodobně značně zpomalí (viz Optimalizace FFTW). Proto je lepší zvětšit překryvnou oblast tak, aby celková velikost subdomény včetně překryvu byla co nejvhodnější pro rychlý výpočet FFT. Negativem tohoto řešení je zvýšený objem komunikace, ta ale tvoří jen malou část výpočetní doby, není tedy tak kritická jako výpočet FFT.
- Druhým způsobem, jak dosáhnout vyšší přesnosti je zvětšení velikosti subdomén snížením jejich počtu. Aby nebylo nutné snižovat počet výpočetních jader, nabízí se použití hybridní verze implementace, ve které jedna subdoména připadá na 8 nebo 16 výpočetních jader.

Hybridní varianta je vhodným řešením i v případě, kdy by jednotlivé subdomény byly už moc malé a redundance výpočtu tedy velmi vysoká.

7.3.1 Příklad

Vraťme se k příkladu uvedenému na konci kapitoly 7.1, kde počítáme doménu o velikosti 3000^3 na 8000 výpočetních jádrech a víme, že dekompozice je možná. Zbývá tedy zvolit vhodnou topologii dekompozice (tím i velikost subdomény) a velikost překryvné oblasti.

Protože nejvhodnějším tvarem subdomény je krychle, přímo se nabízí použití dekompozice $[20, 20, 20]$. V tom případě bude mít každá subdoména rozměr 150^3. Je-li pro přesnost výpočtu nejmenší přípustná hodnota překryvné oblasti 16, zjistíme, jaká je rychlost výpočtu FFT pro domény o velikosti kolem $(150 + 2 * 16)^3$. Doba výpočtu uvedená na řádku s číslem 182 je nižší než v okolních řádcích, ale pro překryv 15 by byl výpočet ještě rychlejší. Naopak, požadujeme-li přesnější výpočet, použijeme velikost překryvu 21 (subdoména 192^3).

Přesnost výpočtu lze pozitivně ovlivnit i použitím hybridní varianty lokální dekompozice. Zvolíme první možnost – 1 subdoména na 8 jader. Pak dělíme doménu pouze na 1000 subdomén, topologie tedy bude $[10, 10, 10]$, velikost subdomény 300^3. Požadujeme-li opět minimální velikost překryvné oblasti 16, zvolíme raději 18, protože doba výpočtu FFT nad doménou o velikosti 336^3 je oproti doméně 332^3 téměř čtvrtinová a tak i srovnatelná s MPI verzí výpočtu.

Na celkovou dobu výpočtu pak ale bude mít vliv i násobení maticí Bell a výměna dat, tudíž nelze dobu výpočtu FFT chápat jako jediný parametr.

7.4 Budoucí vývoj

Protože se potvrdilo, že lokální Fourierova transformace využívající lokální dekompozici dat má vlastnosti, které byly při jejím návrhu očekávány, neměla by tato metoda upadnout v zapomnění, naopak by mohla nalézt široké uplatnění všude tam, kde je dnes používána globální Fourierova transformace využívající globální dekompozici dat.

V MPI verzi simulačního programu k-Wave by třída `Matrix`, nebo alespoň některé její metody, používající lokální dekompozici mohla po úpravách zaujmout místo dosud používané třídy `TMPI_FFTWComplexMatrix`. Zásadní rozdíl je pouze v tom, že v dosavadní verzi je matice po vykonání Fourierovy transformace uložena transponovaná (Y⇔Z) a při lokální dekompozici nikoli a bylo by nutné ve všech metodách pracujících s komplexním výstupem FFT změnit indexy. Dalším větším rozdílem je způsob uložení matic, kdy lokální dekompozice používá pro uložení reálné a komplexní matice dvě různá pole. Poslední větší nutnou změnou v kódu by bylo vyvolání výměny dat mezi subdoménami.

Při použití v programu k-Wave by mohlo dojít až k několikanásobnému zkrácení doby výpočtu oproti stávajícímu řešení.

Jako další krok se nabízí rozšíření mezi širší veřejnost tak, aby mohl tuto implementaci Fourierovy transformace využívající lokální dekompozici dat použít každý, kdo pro ni najde vhodné využití. Nabízí se publikování zdrojových kódů na internetu například na serveru Github[33] nebo podobném, případně vytvoření vlastního webu. Pro veřejné publikování je otázkou, zda dále implementovat další metody pro například aritmetické operace nad maticí či implementovat podporu datového typu `double`. V současné podobě jsou zdrojové kódy publikovatelné.

[33] Github, www.github.com

8 Závěr

Cílem této diplomové práce bylo navrhnout a realizovat nový způsob výpočtu Fourierovy transformace využívající lokální dekompozici dat, toto nové řešení otestovat a porovnat se v současnosti používanou globální metodou dekompozice či jinými, pokud existují, zhodnotit výsledky a v případě úspěchu nového řešení navrhnout jeho další použití. Hlavní motivací pro vytvoření tohoto nového řešení je jeho možné zabudování do simulačního programu k-Wave, od čehož si jeho autoři slibují mimo jiné především zrychlení výpočtu.

Základnímu seznámení se simulačním programem k-Wave se věnovala kapitola 2, v jejíchž podkapitolách byly nejdříve vyjmenovány klíčové funkce programu, následně byla popsána stávající distribuovaná implementace používající globální 1D dekompozici dat. Nakonec byly diskutovány limity a problémy této stávající implementace – především značný objem komunikace mezi procesy a nemožnost použití většího počtu procesorů, než je nejdelší rozměr domény. V této kapitole bylo i okrajově diskutováno použití 2D dekompozice, která sice již netrpí omezením počtu procesorů na nejdelší rozměr, ale stále pro své fungování vyžaduje transpozici celé domény, tudíž objem komunikace neklesá, naopak strmě stoupá.

Následující kapitoly se již zabývají především Fourierovou transformací využívající lokální dekompozici dat. Nejprve je uveden návrh metody, její očekávané vlastnosti a omezení, následně je popsán způsob, jakým byla metoda implementována a testována, nakonec jsou uvedeny a diskutovány výsledky měření a testování.

Ze změřených výsledků vyplývá, že lokální 3D dekompozice dat může být pro výpočet Fourierovy transformace z hlediska rychlosti výpočtu klidně i třikrát lepší řešení, než globální 1D dekompozice. Negativem lokální dekompozice dat je jistá nepřesnost výpočtu, která však při správném použití této nové metody může být velmi malá až zanedbatelná.

V dřívějších letech by byla i drobná chyba výpočtu požadována za nepřípustnou, v dnešní době se ale trend obrací a výměnou za rychlost výpočtu je tolerována i jeho drobná nepřesnost, která však nesmí přesáhnout dané meze. Výpočet FFT využívající lokální dekompozici dat by tedy mohl najít uplatnění v mnoha oborech, ve kterých není požadována absolutní přesnost. Ukáže až praxe, jak se toto nové řešení bude chovat při použití v podmínkách reálného světa, které nelze při vývoji předpokládat, a tedy ani testovat.

Jako příklad výše uvedeného lze zmínit možné začlenění do simulačního programu k-Wave. Může se ukázat, že pro malé velikosti překryvu není výpočet dostatečně přesný, a při dosažení požadované přesnosti již nebude dostatečně rychlý. Stejně tak ale může tato obava být naprosto lichá a metoda výpočtu Fourierovy transformace využívající lokální dekompozici dat bude výrazným příspěvkem ke zrychlení tohoto programu. Vytvoření metody výpočtu FFT využívající lokální dekompozici dat je jen jedním z dílků mozaiky, souběžně s vývojem této metody jsou zkoumány možnosti paralelizace výpočtu v programu k-Wave pomocí svazku grafických karet či výpočetních akcelerátorů Intel Xeon Phi.

Mám-li zmínit přínos, který měla tato práce přímo pro moji osobu, musím jmenovat především novou zkušenost s prací na mnohonásobně větším projektu, než na jakých jsem měl do této doby možnost pracovat. Dalším obohacením bylo zjištění, že při vývoji takto rozsáhlých projektů se ukazují jistá specifika komunikace, která se u menších projektů neprojevují a o nichž teorie mlčí.

Literatura

1. **Jaroš, Jiří, Rendell, Alistair P. a Treeby, Bradley E.** Full-wave nonlinear ultrasound simulation on distributed clusters with applications in high-intensity focused ultrasound. *The International Journal of High Performance Computing Applications.* 2015, Sv. 2, stránky 1-19.

2. k-Wave: A MATLAB toolbox for the time-domain simulation of acoustic wave fields. *k-Wave.org.* [Online] 10. listopad 2014. [Citace: 12. leden 2015.] http://www.k-wave.org/.

3. **Treeby, Bradley E. a Cox, Ben T.** k-Wave: MATLAB toolbox for the simulation and reconstruction of photoacoustic wave fields. *Journal of Biomedical Optics.* 1. březen 2010, Sv. 15, 2, stránky 021314-021314-12.

4. **Treeby, Bradley E., Cox, Ben T. a Jaroš, Jiří.** k-Wave User Manual. *k-Wave.org.* [Online] 15. listopad 2012. http://www.k-wave.org/manual/k-wave_user_manual_1.0.1.pdf.

5. Compute nodes. *IT4I Docs.* [Online] květen 2013. [Citace: 10. leden 2015.] https://docs.it4i.cz/anselm-cluster-documentation/compute-nodes.

6. Network. *IT4I Docs.* [Online] květen 2013. [Citace: 10. leden 2015.] https://docs.it4i.cz/anselm-cluster-documentation/network.

7. Software. *IT4I Docs.* [Online] květen 2013. [Citace: 10. leden 2015.] https://docs.it4i.cz/anselm-cluster-documentation/software.

8. Accessing the Cluster. *IT4I Docs.* [Online] květen 2013. [Citace: 10. leden 2015.] https://docs.it4i.cz/anselm-cluster-documentation/accessing-the-cluster.

9. MATLAB - The Language of Technical Computing. *MathWorks.com.* [Online] 2014. [Citace: 10. leden 2015.] http://www.mathworks.com/products/matlab/.

10. History of HDF Group. *The HDF group.* [Online] 16. květen 2011. [Citace: 10. leden 2015.] http://www.hdfgroup.org/about/history.html.

11. HDF5 Documentation. *The HDF Group.* [Online] 13. listopad 2014. [Citace: 10. leden 2015.] http://www.hdfgroup.org/HDF5/doc/index.html.

12. FFTW Documentation. [Online] [Citace: 10. leden 2015.] http://www.fftw.org/#documentation.

13. Multi-Dimensional DFTs of Real Data. *FFTW Documentation.* [Online] [Citace: 10. leden 2015.] http://www.fftw.org/fftw3_doc/Multi_002dDimensional-DFTs-of-Real-Data.html.

14. GCC Releases. *GNU Foundation.* [Online] 19. prosinec 2014. [Citace: 10. leden 2015.] https://gcc.gnu.org/releases.html.

15. Open MPI: Open Source High Performance Computing. *Open-MPI.org.* [Online] 20. prosinec 2014. [Citace: 10. leden 2015.] http://www.open-mpi.org/.

16. OpenMP Compilers. *OpenMP.org.* [Online] 8. leden 2015. [Citace: 12. leden 2015.] http://openmp.org/wp/openmp-compilers/.

17. OpenMP Specifications. *OpenMP.org.* [Online] 23. červenec 2013. [Citace: 10. leden 2015.] http://openmp.org/wp/openmp-specifications/.

18. Czech Republic National Supercomputer Center Selects Powerful SGI Supercomputer to Drive Advanced Scientific Research. *sgi.com.* [Online] 18. listopad 2014. [Citace: 11. leden 2015.] http://www.sgi.com/company_info/newsroom/press_releases/2014/november/it4i.html.

19. ARM-based platforms. *Mont Blanc, European approach towards energy efficient high performance.* [Online] 2014. [Citace: 13. leden 2015.] http://www.montblanc-project.eu/arm-based-platforms.

20. **Israeli, M., Vozovoi, L. a Averbuch, A.** Spectral multidomain technique with Local Fourier Basis. *Journal of Scientific Computing.* 1993, Sv. 2, 8, stránky 135-149.

21. Real-data DFTs. *FFTW Documentation.* [Online] [Citace: 18. květen 2015.] http://www.fftw.org/doc/Real_002ddata-DFTs.html.

Seznam příloh

Příloha 1 – Matlab kód (chyba vs. překryv)

```
% author      - Bradley Treeby
% date        - 6th July 2014
% last update - 6th July 2014

% setting to plot simulations
plot_sim = false;

% =========================================================================
% SIMULATION CONDITIONS
% =========================================================================

% grid parameters
Nx = 512;
dx = 1e-3;
num_domains = 2;
return_error = true;

% medium properties
c = 1500;
rho0 = 1000;

% calculate an appropriate value for the the time step
cfl = 0.3;
dt = cfl*dx/max(c);

% set the number of time steps
T = 10e-5;
Nt = round(T/dt);

% create impulsive initial pressure, just to the left of the division
p0 = zeros(Nx,1);
p0(Nx/2 - 30) = 1;

% smooth the pressure distribution
win = getWin(Nx, 'Blackman', 'Symmetric', true);

% apply the filter
p0 = real(ifftn(fftn(p0).*ifftshift(win)));

% restore the magnitude to 1
p0 = p0 / max(p0(:));

% =========================================================================
% RUN SIMULATION
% =========================================================================

% set array of overlap sizes to test
overlap_array = 2:19;

% preallocate arrays
error_L2 = zeros(3, length(overlap_array));
error_Linf = zeros(3, length(overlap_array));

% load optimal bell
load bell_shapes;

for index = 1:length(overlap_array)

    % assign overlap size
    overlap = overlap_array(index);

    % erf function
    L = 2;
    bell_x = -1:2/(overlap-1):1;
    epsilon = erf(L * bell_x ./ sqrt( 1 - bell_x.^2 ));
    bell_func = 0.5*(1 + epsilon);

    % run simulation
    output = kspaceFirstOrder1D_decomp(p0, c, rho0, Nx, dx, Nt, dt, num_domains, overlap,
bell_func, plot_sim, return_error);

    % save error
    error_L2(1, index)  = output.L2;
    error_Linf(1, index) = output.Linf;

    % ---------------
```

```
    % Leonid's function
    n = 6;
    bell_func = makeSinBell(overlap, n);

    % run simulation
    output = kspaceFirstOrder1D_decomp(p0, c, rho0, Nx, dx, Nt, dt, num_domains, overlap,
bell_func, plot_sim, return_error);

    % save error
    error_L2(2, index)  = output.L2;
    error_Linf(2, index) = output.Linf;

    % ---------------

    % optimal function
    bell_func = bell_shapes(overlap, 1:overlap);

    % run simulation
    output = kspaceFirstOrder1D_decomp(p0, c, rho0, Nx, dx, Nt, dt, num_domains, overlap,
bell_func, plot_sim, return_error);

    % save error
    error_L2(3, index)  = output.L2;
    error_Linf(3, index) = output.Linf;

end

% ============================================================================
% PLOT
% ============================================================================

figure;
semilogy(overlap_array, error_L2);
ylabel('L2 error');
xlabel('Overlap Size');
legend('erf', 'sin', 'opt');
grid on;

figure;
subplot(2, 1, 1)
semilogy(overlap_array, error_L2);
ylabel('L2 error');
xlabel('Overlap Size');
legend('erf', 'sin', 'opt');
grid on;

subplot(2, 1, 2);
semilogy(overlap_array, error_Linf);
ylabel('L_{\infty} error');
xlabel('Overlap Size');
legend('erf', 'sin', 'opt');
grid on;
```

Příloha 2 – Matlab kód (chyba vs. doména)

```
% author      - Bradley Treeby
% date        - 6th July 2014
% last update - 6th July 2014

% this example shows that the error doesn't change much with the overall
% size of the domain.

% =========================================================================
% CREATE THE COMPUTATIONAL GRID
% =========================================================================

% decomposition parameters
dx = 1e-3;
num_domains = 2;
overlap = 16;

% medium properties
c    = 1500;
rho0 = 1000;

% calculate an appropriate value for the the time step
dt = 0.3*dx/max(c);

% set number of time steps
Nt = 100;

% other settings
bell_func = [];
plot_sim = true;
return_error = true;

% =========================================================================
% RUN SIMULATION
% =========================================================================

% grid sizes to test
Nx_array = 128:64:1024;

% preallocate arrays
error_L2 = zeros(length(Nx_array), 1);
error_Linf = zeros(length(Nx_array), 1);

for Nx_index = 1:length(Nx_array)

    % assign grid size
    Nx = Nx_array(Nx_index);

    % create impulsive initial pressure, just to the left of the division
    p0 = zeros(Nx,1);
    p0(Nx/2 - 15) = 1;

    % smooth the pressure distribution
    win = getWin(Nx, 'Blackman', 'Symmetric', true);

    % apply the filter
    p0 = real(ifftn(fftn(p0).*ifftshift(win)));

    % restore the magnitude to 1
    p0 = p0 / max(p0(:));

    % run simulation
    output = kspaceFirstOrder1D_decomp(p0, c, rho0, Nx, dx, Nt, dt, num_domains, overlap,
bell_func, plot_sim, return_error);

    % save error
    error_L2(Nx_index)  = output.L2;
    error_Linf(Nx_index) = output.Linf;

end
% =========================================================================
% PLOT
% =========================================================================

figure;
plot(Nx_array, error_L2, 'b-');
ylabel('L2 error');
xlabel('grid size');
```

Příloha 3 – Matlab kód (výpočet Laplaciánu)

```
% Spectral derivatives using local Fourier basis with domain decomposition
% in all three dimensions.
%
% Date: 12th November 2014
% Last Update: 30th November 2014
% Author: Bradley Treeby

% define grid parameters
Nx_full = 128;
Ny_full = Nx_full;
Nz_full = 256;
dx = 0.1;
dy = 0.1;
dz = 0.1;

% calculate the global wavenumbers
kgrid_full = makeGrid(Nx_full, dx, Ny_full, dy, Nz_full, dz);

% create the input function (this could be anything)
f = bsxfun(@times, reshape(getWin(Nz_full, 'Hanning'), [1, 1, Nz_full]),
repmat(reshape(peaks(Nx_full), [Nx_full, Nx_full, ]), [1, 1, Nz_full]));


% compute the Laplacian spectrally in the normal way
tic;

dfdx = real(ifftn( fftshift(1i*kgrid_full.kx).* fftn(f) ));
dfdy = real(ifftn( fftshift(1i*kgrid_full.ky).* fftn(f) ));
dfdz = real(ifftn( fftshift(1i*kgrid_full.kz).* fftn(f) ));

disp(['Global gradient calculated in ' scaleSI(toc) 's'] );
```

Příloha 4 – Doba výpočtu Laplaciánu

Domain	1 Core	1 Socket	1 Node
64	0,01393	0,002082	0,001561
65	0,029376	0,007411	0,003567
66	0,028857	0,004758	0,003206
67	0,101295	0,019935	0,017004
68	0,044097	0,007039	0,004799
69	0,058783	0,011659	0,00518
70	0,032873	0,007776	0,012076
71	0,100255	0,019839	0,010971
72	0,024231	0,004025	0,003012
73	0,153133	0,028081	0,015256
74	0,082528	0,01256	0,008626
75	0,040508	0,00645	0,003916
76	0,062935	0,011637	0,006639
77	0,052671	0,009315	0,004761
78	0,048994	0,007171	0,007429
79	0,146242	0,100448	0,033673
80	0,029157	0,005474	0,002947
81	0,05617	0,014124	0,107631
82	0,119281	0,017697	0,010615
83	0,199819	0,035455	0,04468
84	0,118198	0,011108	0,004581
85	0,09728	0,015665	0,019261
86	0,143317	0,021103	0,016462
87	0,134421	0,026957	0,013401
88	0,052759	0,016464	0,004476
89	0,248907	0,060409	0,041665
90	0,065189	0,009411	0,006932
91	0,085414	0,012674	0,007453
92	0,123965	0,022909	0,014273
93	0,17316	0,032535	0,015712
94	0,191206	0,035848	0,093928
95	0,143528	0,02711	0,011024
96	0,051442	0,00758	0,005527
97	0,337828	0,060691	0,06467
98	0,096549	0,017727	0,010691
99	0,111125	0,023673	0,013106
100	0,067879	0,010728	0,006159
101	0,392923	0,066423	0,501411
102	0,148106	0,021774	0,013253
103	0,491443	0,072207	0,064211
104	0,08649	0,013535	0,007274
105	0,11944	0,026655	0,016909
106	0,31212	0,045475	0,028473
107	0,47621	0,087543	0,070414
108	0,087046	0,014273	0,00834
109	0,730724	0,108031	0,091573
110	0,145545	0,020772	0,011429
111	0,324766	0,056993	0,038468
112	0,10078	0,015508	0,012214
113	0,748568	0,127602	0,087319
114	0,293472	0,032986	0,024861
115	0,283669	0,055359	0,045685
116	0,285743	0,045512	0,034633
117	0,192572	0,032433	0,019651

Domain	1 Core	1 Socket	1 Node
118	0,454388	0,072648	0,115354
119	0,280565	0,052786	0,025682
120	0,117333	0,017861	0,01153
121	0,173285	0,046485	0,018845
122	0,531793	0,079124	0,051335
123	0,483632	0,084129	0,039288
124	0,3575	0,05761	0,031849
125	0,216845	0,03798	0,02499
126	0,289715	0,031559	0,016996
127	0,791051	0,271151	0,092606
128	0,11629	0,017032	0,010367
129	0,566397	0,104717	0,086562
130	0,320011	0,042538	0,026737
131	1,28345	0,22156	0,15913
132	0,205311	0,032112	0,017237
133	0,411887	0,074435	0,033467
134	0,825093	0,108303	0,059092
135	0,268942	0,062229	0,022681
136	0,348691	0,051662	0,038587
137	1,1436	0,181132	0,302082
138	0,440095	0,065997	0,039673
139	1,11852	0,218691	0,152764
140	0,298427	0,036058	0,025655
141	0,769716	0,148623	0,206155
142	0,58007	0,088271	0,08849
143	0,359909	0,22466	0,045867
144	0,166033	0,027041	0,017885
145	0,650912	0,18689	0,177703
146	1,10587	0,310752	0,082809
147	0,421301	0,073305	0,038852
148	0,682109	0,107993	0,056225
149	1,59234	0,256388	0,171505
150	0,357049	0,056146	0,038938
151	1,45414	0,245166	0,301255
152	0,567788	0,085329	0,07146
153	0,693027	0,112927	0,053052
154	0,461659	0,069542	0,044522
155	0,826309	0,149986	0,301626
156	0,341149	0,053125	0,035769
157	1,7479	0,372783	0,216068
158	0,884046	0,132029	0,262852
159	1,23972	0,357158	0,262716
160	0,240292	0,039934	0,022629
161	0,817266	0,151898	0,122286
162	0,500519	0,090015	0,049226
163	2,12369	0,376304	0,260652
164	1,04822	0,154732	0,088469
165	0,617901	0,087406	0,062376
166	1,30875	0,34419	0,288466
167	2,3634	0,490593	0,456358
168	0,379581	0,058025	0,047562
169	0,743591	0,230831	0,0582
170	0,727765	0,109334	0,069291
171	1,01876	0,256595	0,073488

Domain	1 Core	1 Socket	1 Node
172	1,19756	0,190371	0,11423
173	2,67224	0,419317	0,35212
174	1,00245	0,14386	0,090774
175	0,645675	0,13354	0,103762
176	0,475294	0,070162	0,131802
177	1,93816	0,338444	0,459235
178	1,50386	0,375913	0,348231
179	2,87333	0,496631	0,278754
180	0,581812	0,084802	0,041959
181	3,28422	0,513734	0,270346
182	0,896841	0,264252	0,075981
183	2,19099	0,375948	0,17692
184	1,15904	0,157105	0,090235
185	1,67091	0,280864	0,142676
186	1,30765	0,204121	0,102372
187	1,23866	0,213171	0,104564
188	1,74451	0,293524	0,821204
189	1,12412	0,199174	0,269174
190	1,15774	0,167274	0,086657
191	3,07739	0,50237	0,358072
192	0,485259	0,078872	0,052067
193	2,83543	0,589196	0,412837
194	1,96425	0,303605	0,460214
195	1,06903	0,187487	0,106342
196	0,776152	0,118948	0,075138
197	3,12429	0,597732	0,420817
198	1,0255	0,144924	0,077784
199	3,20645	0,671821	0,37696
200	0,618059	0,102763	0,056832
201	3,13299	0,665649	0,297473
202	2,31811	0,354158	0,453316
203	1,99315	0,457891	0,208787
204	1,25446	0,184927	0,104675
205	2,41467	0,423552	0,278262
206	2,37827	0,361428	0,523713
207	1,84889	0,348286	0,230067
208	0,881565	0,130046	0,077517
209	1,90281	0,520902	0,159404
210	1,13998	0,197631	0,087323
211	3,84117	0,759179	0,425714
212	2,73782	0,396964	0,221609
213	3,51041	0,699344	0,316083
214	2,68502	0,396987	0,533264
215	2,87878	0,473428	0,39931
216	0,965688	0,151509	0,084094
217	2,40821	0,482459	0,207494
218	4,74996	0,724354	0,451388
219	4,6492	0,754096	0,523315
220	1,13149	0,176555	0,274144
221	2,09901	0,387256	0,224034
222	2,53741	0,351415	0,287454
223	7,37098	1,275	0,840426
224	0,954292	0,1458	0,09041
225	1,50895	0,478122	0,142764

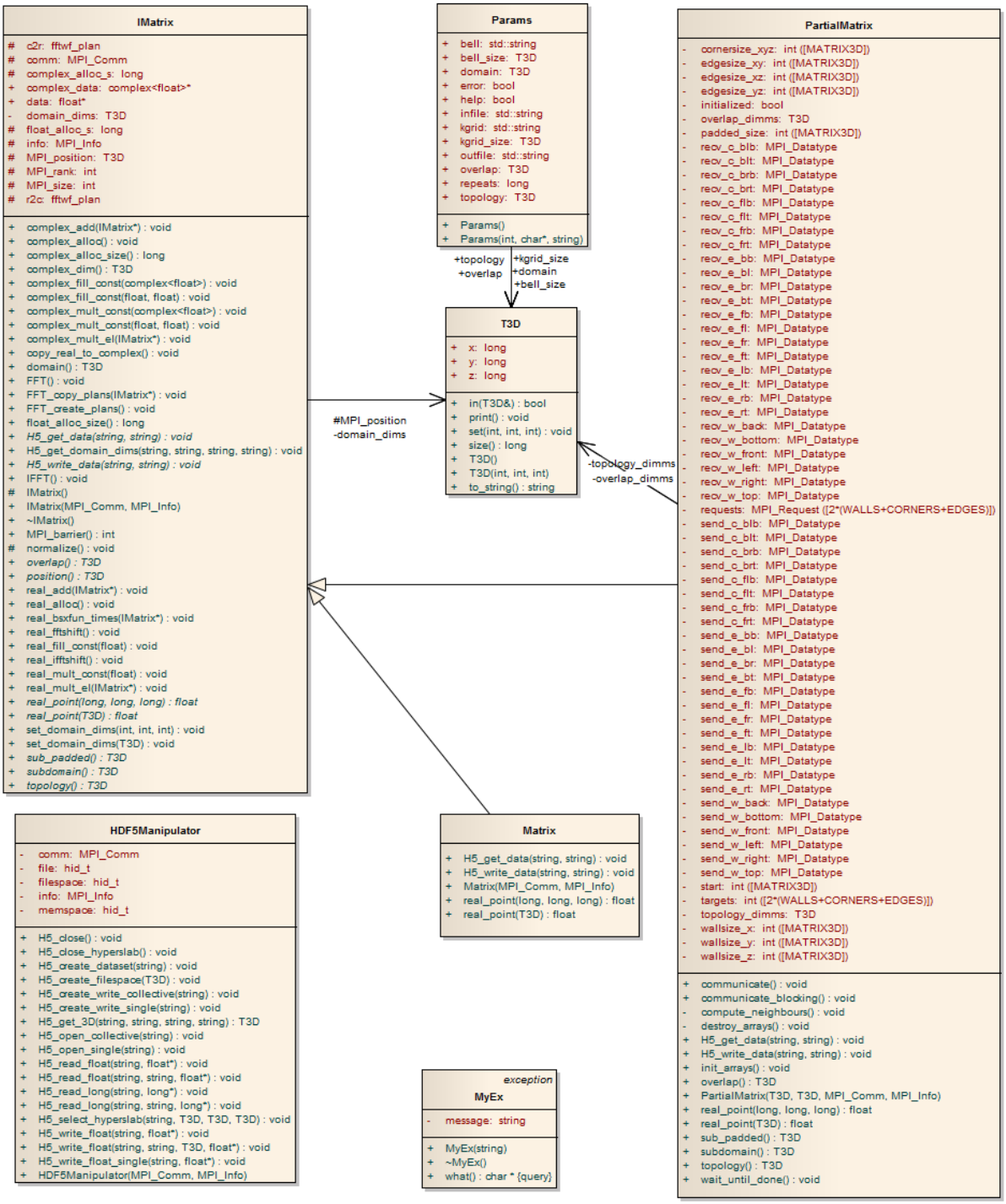
Domain	1 Core	1 Socket	1 Node
226	5,26877	0,741401	0,450514
227	5,56696	0,973404	1,1483
228	2,01556	0,325566	0,199041
229	5,83251	1,04805	0,816192
230	2,34679	0,330422	0,337579
231	1,92395	0,360423	0,183958
232	2,50477	0,352054	0,20877
233	6,09315	1,11612	0,919022
234	1,88834	0,261716	0,223473
235	4,00092	0,908129	1,13026
236	3,92978	0,755704	0,939379
237	4,83278	0,822476	1,21166
238	2,33944	0,35849	0,201472
239	6,34201	1,15909	0,903195
240	1,05514	0,16715	0,115549
241	7,84981	1,42152	0,903066
242	1,79657	0,410246	0,189161
243	2,29171	0,400732	0,519681
244	4,68494	0,688128	0,365314
245	2,60651	0,538588	0,234267
246	3,56489	0,561661	0,271176
247	2,89118	0,971994	0,37945
248	3,19034	0,450724	0,259198
249	6,54633	1,06116	0,936561
250	1,91871	0,316984	0,310105
251	6,31831	1,29713	0,573008
252	1,80721	0,412133	0,166366
253	3,41445	0,625126	0,281296
254	3,37347	0,610125	0,719946
255	3,04458	0,461732	0,273202
256	1,19569	0,292896	0,126236
257	7,41228	1,48323	0,915937
258	4,35389	0,702609	0,377326
259	4,74853	0,948232	0,468
260	1,79709	0,263631	0,154637
261	4,34817	0,853851	0,436442
262	8,89501	1,24051	1,01468
263	9,38612	1,5806	1,34998
264	2,02278	0,296959	0,570476
265	6,14455	1,15521	0,61711
266	3,53924	0,681529	0,349586
267	8,10235	1,47447	1,35677
268	6,52533	1,1218	0,648726
269	10,0117	1,72322	1,68746
270	2,68685	0,365912	0,204553
271	8,40213	1,56432	0,98779
272	3,18761	0,541021	0,24789
273	3,59871	0,817725	0,279339
274	4,95582	0,80906	0,96913
275	3,30326	0,558261	0,284232
276	4,11021	0,619705	0,380228
277	10,555	1,88239	1,1078
278	4,90076	0,903577	0,95258
279	5,38344	1,13434	0,647288
280	2,17095	0,360463	0,253459
281	9,40255	1,61011	0,876645
282	5,93934	1,05539	1,64193
283	10,8444	1,79874	1,52865
284	5,15406	0,767915	0,663484
285	4,79167	0,807642	0,386735
286	2,69072	0,558998	0,271215
287	6,7167	1,2312	0,943611
288	1,83356	0,289645	0,195145
289	6,19825	1,84226	0,914975
290	5,13362	0,778215	0,452904
291	87,6884	1,91359	2,01112
292	76,0869	1,45343	1,05174
293	84,1849	2,16641	1,55991
294	14,1087	0,496251	0,274847
295	23,9923	1,7871	1,80356
296	11,947	0,867097	0,486857
297	9,40839	0,997503	0,383023
298	6,92575	1,06015	1,12586
299	7,92728	1,16133	0,482745
300	5,32109	0,515666	0,232275
301	16,9687	1,62055	1,13193
302	11,1515	0,89341	1,12824
303	23,9215	2,05027	2,12986
304	10,5588	0,765639	0,387744
305	21,3265	1,93328	1,02649
306	10,041	0,899313	0,449678
307	38,9655	3,32986	2,16878
308	6,71593	0,503117	0,343367
309	27,572	2,42177	1,50646
310	12,9093	0,955012	0,683367
311	40,6046	3,45537	2,81197
312	5,60799	0,477209	0,393386
313	38,5698	3,36827	2,15017
314	12,2223	0,994338	1,25194
315	9,7205	0,763254	0,655638
316	17,5084	1,35541	2,2751
317	30,3078	2,73694	1,80895
318	17,3866	1,34432	0,682868
319	16,1741	1,16463	1,02077
320	4,97454	0,406386	0,351551
321	27,9048	2,79693	2,84256
322	6,73845	1,13669	0,547895
323	8,84388	1,86828	1,38098
324	3,76679	0,555759	0,348399
325	5,81539	0,86568	0,746998
326	10,0912	1,6548	1,33225
327	19,7213	3,34338	2,05975
328	8,411	1,29724	0,985494
329	11,4341	2,14497	2,93251
330	17,6599	0,902046	0,556938
331	149,702	4,0484	2,60678
332	87,1673	1,90621	2,43873
333	10,0551	1,9611	1,2799
334	9,97486	1,49399	1,49729
335	14,4712	2,86338	2,7196
336	3,2358	0,490776	0,311027
337	23,7684	4,37597	3,01849
338	5,35072	1,06055	0,555201
339	22,2966	3,96937	2,4686
340	20,7807	0,950081	0,495144
341	54,3118	2,00696	1,06445
342	57,5566	1,16632	1,16201
343	53,6113	1,52585	0,74917
344	89,2661	1,69503	1,0847
345	9,65777	1,83524	1,11463
346	10,5537	1,7444	1,74817
347	20,173	3,66596	2,92102
348	8,26533	1,39091	0,871841
349	20,8055	3,62418	2,95923
350	21,8283	0,876586	0,466367
351	27,8807	1,15373	0,831115
352	31,867	0,667394	0,382429
353	152,635	3,62704	3,00982
354	14,356	2,64787	3,11527
355	15,8676	3,09639	2,2953
356	12,384	2,18413	2,76841
357	8,934	1,65868	0,934994
358	11,276	1,87285	1,65402
359	48,655	3,72696	2,7675
360	17,1496	0,659979	0,514612
361	81,9071	3,10616	1,48879
362	174,248	3,15479	1,97888
363	24,2692	1,28132	0,814244
364	17,0219	1,06255	0,479553
365	66,6817	4,04686	2,83979
366	47,3238	2,31052	1,49795
367	100,636	5,90585	4,40905
368	28,6127	1,4408	0,832633
369	53,4773	2,90175	1,57799
370	93,2837	1,92658	1,43413
371	141,247	3,12178	2,00146
372	66,0533	1,72369	1,23163
373	106,494	6,33003	3,81241
374	28,051	1,52006	0,839532
375	24,8625	1,76108	1,10507
376	39,8535	2,74323	5,65143
377	41,7979	2,63061	1,09241
378	27,9566	1,38785	1,0403
379	44,2651	3,85624	2,63288
380	70,8223	1,50809	0,904219
381	197,597	3,99186	3,22547
382	46,3949	1,70089	1,81359
383	74,3935	4,11913	3,43571
384	14,5583	0,977986	0,426355
385	31,2915	2,13108	0,985438
386	37,6934	2,04521	2,80416
387	50,6066	3,26918	2,86906
388	46,9979	2,65341	3,34589
389	90,269	4,70522	3,67058
390	69,9318	1,39462	0,745361
391	138,409	4,19347	1,58079
392	57,0926	1,20728	0,840565
393	81,6472	6,10806	4,4687
394	11,8303	1,91956	1,95058
395	22,2383	3,92534	3,70783
396	7,32354	1,36967	0,644399
397	27,1894	4,87339	3,85902
398	12,238	1,94562	2,05959
399	13,1735	2,55618	2,1899

Domain	1 Core	1 Socket	1 Node
400	5,272	0,866803	0,63311
401	37,1691	6,19692	4,24648
402	22,69	3,52537	2,17615
403	16,9128	3,33565	2,15528
404	19,7102	3,4292	3,62927
405	11,1965	1,94995	1,38729
406	15,11	2,28014	1,69277
407	18,6139	3,75995	2,07269
408	11,0671	1,7901	0,983309
409	34,0555	5,84336	3,83044
410	17,7547	2,78099	1,80714
411	33,3839	5,83868	4,77734
412	20,4073	3,30167	4,27711
413	27,0763	5,12597	3,81559
414	14,9388	2,25928	1,15091
415	29,5242	5,27772	4,14429
416	7,31896	1,15932	0,742714
417	35,2967	6,0643	6,10573
418	14,4539	2,42616	1,65239
419	36,6394	6,4454	4,50926
420	8,06854	1,18833	0,877335
421	32,1356	5,82169	4,40343
422	17,122	2,79649	2,27559
423	23,8991	4,58864	4,5319
424	24,1304	3,61545	2,26586
425	16,9263	3,36645	1,43803
426	18,4104	2,97572	2,22862
427	29,1175	5,28989	2,67427
428	42,897	3,78666	4,04911
429	13,8008	2,50691	1,95809
430	21,0312	3,09248	2,08203
431	39,0754	6,45505	5,35611
432	7,26884	1,0986	0,721303
433	37,9044	6,82964	3,43403
434	19,0312	2,83414	2,01076
435	20,7342	3,85065	2,74901
436	39,4691	6,08455	3,9177
437	24,8578	3,91882	2,10859
438	31,905	4,76033	3,04849
439	45,038	7,91325	5,03237
440	9,50693	1,51593	0,966004
441	14,4999	2,47558	1,45579
442	15,4068	2,59628	1,65066
443	44,899	8,13156	4,40485
444	19,4714	3,04832	2,01848
445	38,0678	6,73871	5,14944
446	43,489	6,28919	3,49094
447	45,0341	7,40344	5,28986
448	8,26487	1,21797	0,718661
449	40,4486	6,9683	6,46435
450	13,8811	2,52	1,56841
451	27,0955	5,25619	4,37623
452	41,5252	6,44491	4,62642
453	43,7415	7,67337	6,7451
454	21,1056	3,34369	2,71177

Domain	1 Core	1 Socket	1 Node
455	15,647	3,41542	1,31505
456	15,4226	2,51183	1,77538
457	46,0588	8,34573	6,43925
458	21,2926	3,60329	2,72449
459	19,4089	3,46897	2,30499
460	17,4826	2,85648	1,90441
461	49,4943	8,58717	6,20923
462	14,9265	2,30329	1,29403
463	42,8733	7,80406	5,67492
464	20,2031	3,07592	1,80728
465	24,3302	4,7857	3,15213
466	21,4337	3,52096	2,86617
467	49,0774	8,92429	6,28739
468	13,5353	2,30739	1,34555
469	42,8525	8,17374	4,49378
470	27,5483	5,17102	11,1394
471	49,4495	8,42489	7,51264
472	34,5383	5,50379	9,26681
473	31,968	6,26371	3,69458
474	24,6116	4,17839	7,07453
475	110,532	4,01217	2,42083
476	94,5836	3,13363	2,06163
477	138,792	7,66056	3,33964
478	96,9345	3,76874	2,98556
479	210,519	9,1331	6,51691
480	44,1366	1,46918	0,948312
481	160,559	9,45371	3,40078
482	196,829	7,6455	5,06861
483	104,132	5,5776	4,69876
484	46,6569	2,43878	1,40221
485	151,303	9,62815	8,4066
486	43,2188	3,05449	1,50341
487	52,6283	9,59304	5,65777
488	39,3424	6,48715	3,41962
489	63,5783	11,6142	7,68647
490	108,657	3,52433	2,11704
491	55,0258	9,77668	5,81003
492	28,2501	4,4919	2,74936
493	36,3946	6,95908	4,74828
494	21,4006	3,45648	1,70231
495	20,0348	3,68947	2,31249
496	25,1613	4,01949	2,60297
497	45,9293	7,92046	5,02066
498	38,3959	6,10301	8,16969
499	65,3501	11,5368	6,64922
500	66,6865	2,33022	1,29948
501	189,635	11,36	6,34684
502	24,96	4,36666	3,61555
503	80,028	9,90447	6,48498
504	25,6408	2,30992	1,32033
505	117,711	9,93442	7,11767
506	116,743	4,65796	2,6757
507	97,4112	3,48001	2,66712
508	110,956	4,55913	5,70079
509	188,866	10,2098	7,88409

Domain	1 Core	1 Socket	1 Node
510	55,1408	3,95584	2,17903
511	122,977	9,96583	4,72911
512	24,2165	2,16553	1,29633
513	89,8626	6,21887	3,51613
514	148,086	6,00686	4,15453
515	252,175	11,1565	9,3772
516	152,261	5,75061	3,39943
517	129,421	8,94602	11,0634
518	108,605	5,51561	2,77132
519		14,2662	6,96323
520		2,3213	1,84761
521		12,6762	9,01826
522		5,00749	2,93732
523		13,6954	9,6915
524		11,6791	6,7449
525		4,00015	2,33052
526		5,90491	3,68414
527		8,80932	6,3615
528		2,5384	1,64505
529		8,4732	5,68586
530		7,48672	4,48984
531		10,6755	6,83049
532		4,56663	2,31329
533		7,87129	4,15887
534		7,21193	9,04513
535		13,1705	10,2056
536		9,1773	5,10397
537		15,8402	9,55901
538		6,70513	4,50027
539		4,55733	2,46438
540		3,0387	1,68812
541		16,8875	11,618
542		5,35696	3,96568
543		16,1768	10,5719
544		3,93138	2,23853
545		16,0949	8,87682
546		4,55013	2,96251
547		18,8438	11,9759
548		7,15841	6,97825
549		11,9473	5,95439
550		4,95401	2,3687
551		10,5895	5,05266
552		5,40302	3,52861
553		12,7228	10,8658
554		6,57784	4,60349
555			4,54934
556			7,03174
557			13,6451
558			3,4221
559			8,08678
560			1,80248

Příloha 5 – Diagram tříd



Příloha 6 – Obsah CD

`doc/`	Dokumentace vygenerovaná programem Doxygen
`indata/`	Ukázková vstupní data
`indata/kgrids/`	Matice KGrid pro ukázková vstupní data
`indata/bells/`	Matice Bell pro ukázková vstupní data
`kWave-matlab/`	k-Wave toolbox pro Matlab
`matlab_scripts/`	Skripty pro Matlab (generování vstupních dat)
`measure/`	Výsledky měření
`outdata/`	Výstupní data
`pbs/`	Ukázka PBS skriptu
`reports/`	Ukázkové výstupní zprávy profileru
`shell_scripts/`	Skripty pro linuxový shell (generování dat, spouštění)
`src/`	Zdrojové kódy
`Doxyfile`	
`Makefile`	
`Readme.txt`	Stručný návod k použití, příklady spuštění
`Technická zpráva.pdf`	Technická zpráva ve formátu PDF