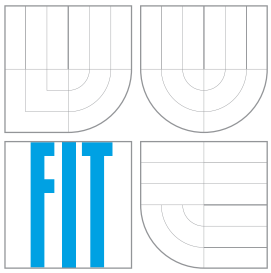
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

# OPTIMALIZACE DISTRIBUOVANÉHO I/O SUBSYSTÉMU PROJEKTU K-WAVE

OPTIMIZATION OF THE DISTRIBUTED I/O SUBSYSTEM OF THE K-WAVE PROJECT

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. ONDŘEJ VYSOCKÝ

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. JIŘÍ JAROŠ, Ph.D.

BRNO 2016

# Abstrakt

Práce se zabývá řešením efektivního paralelního zápisu a čtení dat pro nástroj k-Wave, provádějící simulací šíření ultrazvuku. Tento nástroj je superpočítačovou aplikací, proto je spouštěn na souborovém systému Lustre a vyžaduje paralelní zpracování pomocí MPI a zápis ve formátu vhodném pro velké množství dat (HDF5). V rámci této práce byly navrženy metody efektivního způsobu zápisu dat dle potřeb k-Wave, pomocí kumulace dat a přerozdělování. Všechny metody zrychlily nativní zápis a vedly až k rychlosti zápisu 13,6 GB/s. Popsané metody jsou požitelné pro všechny aplikace s distibuovanými daty a častým zápisem.



# Abstract

This thesis deals with an effective solution of the parallel I/O of the k-Wave tool, which is designed for time domain acoustic and ultrasound simulations. k-Wave is a supercomputer application, it runs on a Lustre file system and it requires to be implemented with MPI and stores the data in suitable data format (HDF5). I designed three methods of optimization which fits k-Wave's needs. It uses accumulation and redistribution techniques. In comparison with the native write, every optimization method led to better write speed, up to 13,6 GB/s. It is possible to use these methods to optimize every data distributed application with the write speed issue.



# Klíčová slova

k-Wave, paralelní programování, MPI, HDF5, Lustre, superpočítač, I/O, zápis, optimalizace

# Keywords

k-Wave, Parallel Programming, MPI, HDF5, Hierarchical Data Format, Lustre, Supercomputer, I/O, Write, Optimization


# Citace

VYSOCKÝ, Ondřej. *Optimalizace distribuovaného I/O subsystému projektu k-Wave*. Brno, 2016. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Jaroš Jiří.

# Optimalizace distribuovaného I/O subsystému projektu k-Wave

## Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Jiřího Jaroše, Ph.D.

.....
Ondřej Vysocký
25. května 2016

## Poděkování

Tato práce byla podporována Ministerstvem školství, mládeže a tělovýchovy z projektu „IT4Innovations National Supercomputing Center – LM2015070“ pro podporu velkých výzkumných infrastruktur, experimentální vývoj a inovativní projekty.

Chtěl bych poděkovat mému vedoucímu práce, Ing. Jiřímu Jarošovi, Ph.D., za jeho čas a vždy velkou ochotu pomoci. Dále děkuji své rodině za poskytnutí pevného zázemí, které mi umožnilo vystudovat tuto školu. Především však chci poděkovat moji přítelkyni Nicole Kociánové za její podporu a trpělivost v průběhu mého celého studia.

# Obsah

# Kapitola 1

# Úvod

Rychlost vývoje pevných disků výrazně zaostává za vývojem procesorů [29], tím pádem se I/O stává úzkou částí a způsobuje výrazné zpomalení celé aplikace. Proto je nutné systém čtení a zápisu vyladit pro maximální zmenšení rozdílu mezi vstupně-výstupní a výpočetní částí. S tímto problémem se setkává mnoho paralelních aplikací, které generují obrovské množství dat a je pro ně proto nutné tento problém řešit.

Jednou z těchto aplikací je nástroj k-Wave [36], který slouží k simulaci šíření ultrazvuku, čehož se využívá především v medicíně nebo při zkoumání materiálů. Z důvodu vysoké výpočetní a paměťové složitosti těchto simulací musí výpočet probíhat na superpočítačích. Prostředí počítačových clusterů je značně specifické, především z důvodů sdílení prostředků s ostatními uživateli, dávkového zpracovávání úloh nebo z dúvodu, který je pro tento případ zásadní, a tím je zápis dat na RAID disková pole se souborovým systémem lustre. V této práci se zabývám MPI verzí nástroje k-Wave, která zapisuje výsledky simulace do HDF5 souboru.

Vstupně-výstupní systém na superpočítačích je tvořen několika vrstvami, které nabízí mnoho nastavitelných parametrů, které ovlivňují zásadním způsobem výkon. Vhodným nastavením těchto parametrů je možné dosáhnout maximální možné rychlosti I/O, avšak jen pro dostatečně velké množství dat. Problémem je však rychlost častého zápisu malého množství dat, jelikož přístup k pevným diskům je v tomto prostředí spojen s velkou režií, která tak zcela degraduje rychlost malých paralelních zápisů.

Rychlost zápisu je pro k-Wave zásadní, jelikož stejně jako ve většině simulačních nástrojů jsou v průběhu simulace vygenerovány až stovky gigabajtů dat, které je nutno zapsat do souboru pro následnou analýzu výsledků. Ač je zapisováno velké množství dat, jejich rozložení mezi iterace simulace a distribuce zapisovaných dat na jednotlivé procesy však může způsobit nízkou rychlost zápisu, která zdaleka nedosahuje maximální možné rychlosti. Spolu s pomalým zápisem roste také doba celé simulace, a tedy i její cena.

Zápis takovýchto souborů je časově velmi náročný a vyžaduje proto optimalizaci. Tato práce se zabývá metodami optimalizace paralelního zápisu, které jsou založeny na přeuspořádání dat mezi procesy a na jejich kumulací s cílem vyrovnat a zvětšit množství dat na zápis.

## 1.1 Motivace

Cílem této práce je vylepšit způsob zápisu dat produkované simulačním nástrojem k-Wave, který stejně jako v mnoha dalších paralelních aplikacích poskytuje velký prostor pro zlepšení. Navržené metody zápisu jsou vyvíjeny a optimalizovány pro aplikaci k-Wave, ale výsledky této práce jsou použitelné pro libovolné paralelní aplikace s distribuovanou pamětí a častým zápisem. Dosažené výsledky jsou srovnány s nativním způsobem zápisu, který využívá k-Wave nyní.

## 1.2 Projekt k-Wave

Nástroj k-Wave [36] slouží pro simulaci šíření akustických a ultrazvukových vln pomocí k-prostorové pseudospektrální analýzy. Projekt k-Wave je mezinárodním projektem, který je v současné době vyvíjen na University College London a Vysokém učením technickém v Brně.

Simulace ultrazvuku mají význam především v medicíně, ale také například pro zkoumání zemského podloží. V medicíně je využití ultrazvuku běžnou záležitostí, díky k-Wave je však nově možné využít ultrazvuku k neinvazivní léčbě nádorů v těle pacienta. K tomuto účelu se je využito vysoce intenzivního ultrazvukového záření (HIFU, High-intensity focused ultrasound), které v ohnisku záření působí zničující silou, čímž dojde k odpaření tkáně. Aby bylo možné záření takto přesně zacílit, je nutné provést zpětnou simulaci šíření ultrazvuku tělem pacienta, která nám dá informaci o vhodném nastavení ultrazvukového zářiče. Vývojářský tým se momentálně zabývá verzi BrainWave, která by měla umožnit simulaci šíření ultrazvuku skrz lebku do mozku pacienta, což by otevřelo nové možnosti operací mozku bez chirurgického zákroku.

Nástroj k-Wave existuje v několika verzích, které se liší podle platformy, na které probíhá výpočet. Nejmenší simulované problémy je možno spouštět v Matlabu, větší simulační domény je možné řešit na GPU nebo MIC. Největší simulace mohou překročit velikost mřížky $2048^3$, přičemž takovéto simulace vedou k vygenerovaným souborům o velikosti 500 GB až 1 TB.

Tato práce optimalizuje vstupně-výstupní rozhraní MPI implementace tohoto nástroje, který je spouštěn na superpočítači Salomon a zapisuje výstup simulace do HDF5 souboru.

## 1.3 Organizace textu

V kapitole o superpočítačích 2 jsou popsány dnešní superpočítače, prostředí v jakém se paralelní aplikace vyvíjí a je zde popsáno jak funguje vstupně-výstupní systém superpočítačů, včetně podrobnějšího popisu ukládání dat na počítači Salomon. Dále jsou zde napsáno o paralelním souborového systému Lustre, a o knihovně HDF5, jíž je využito pro uložení výstupních dat v k-Wave.

V následující kapitole 3 jsou rozebrány jaké existující vzory zápisu v paralelních aplikacích a jaké jsou požadavky na zápis v k-Wave. Následně jsou rozebrány obecné postupy optimalizace zápisu. Na základě těchto postupů je navržena řada způsobů optimalizovaného zápisu, kterým je věnována sekce 3.3. Následuje popis implementace těchto metod v k-Wave a implementace pomocné aplikace provádějící I/O analýzu cílového stroje.

Kapitola 4 uvádí testy, v kterých jsou jednotlivé metody stavěny do situací způsobující pomalý zápis a srovnávají jejich rychlost s rychlostí nativního zápisu v k-Wave.

# Kapitola 2

# Superpočítače

Superpočítače se v dnešní době skládají z velkého množství běžných procesorů. Na jediné základové desce se nachází několik procesorů s vyšším počtem jader. Toto uskupení procesorů nazýváme uzel, přičemž procesory na uzlu sdílí Last Level Cache (LLC), operační paměť a síťovou kartu, která zajišťuje propojení s dalšími uzly a vytváří tak jediný počítačový cluster [12]. U superpočítačů se dbá především na rychlou komunikaci mezi procesory (vedle špičkového síťového vybavení je důležitá také efektivní topologie sítě) a vysoký výkon operací v plovoucí řádové čárce.

Spojení mnoha procesorů pod jediný stroj poskytuje vysoký výpočetní výkon, který je však sdílen mnoha uživateli. Z tohoto důvodu je zavedeno dávkové zpracování, které zajistí spuštění požadovaného programu v momentě, kdy na daný program přijde řada. Říkáme, že vytváříme joby. Jelikož na těchto strojích nebývá jediná fronta, nýbrž několik, které se odlišují dle účelu, priorit, maximální doby běhu jobu nebo povoleného množství alokovaných procesorů, musí existovat program, jež rozhodne, který z čekajících jobů bude spuštěn. Tento software nazýváme Portable Batch System (PBS) [3]. Aby program mohl být zařazen do některé z front, vytváříme bash skripty, které v hlavičce obsahují základní informace pro PBS a dále pak spouštějí požadovaný program. V případě, že byl job vybrán z fronty čekajících ke spuštění, jsou mu přiděleny požadované prostředky, které nesmí přečerpat. Jestliže dojde k přečerpání některého z prostředků, PBS zajistí předčasné ukončení jobu. Spuštěný job má výhradní přístup k procesorům, které mu byly přiděleny, avšak komunikační síť a také I/O subsystém jsou sdíleny s ostatními uživateli, na což musí být brán zřetel.

Vedle dávkového zpracování jobů je na superpočítačích možná také interaktivní práce, používaná především pro zkoušení funkčnosti programu nebo pro spuštění grafických nástrojů. Těmito nástroji mohou být buď ladící programy nebo třeba vizualizace výsledků programu, ať už za běhu nebo post-processingem. V sekcích 2.1.2 a 2.1.3 jsou popsány nástroje pro analýzu paralelních aplikací, které byly při vývoji této práce využity.

V Evropě je většina superpočítačových center spojena pod záštitou neziskové asociace Partnership for Advanced Computing in Europe (PRACE) [24], která zajišťuje spolupráci jednotlivých center a určuje směřování celé komunity. Tato práce je vyvíjena na superpočítači Salomon, který je jedním z počítačů IT4Innovation centra [8], spadajícím pod Vysokou školu báňskou v Ostravě, které je českým zástupcem v PRACE. Salomon nabízí 1 008 výpočetních uzlů, tvořených dvanácti-jádrovými procesory Intel Xeon E5-2680v3, s celkovým teoretickým výkonem 2 011,7 TeraFlopů, s nímž se zařadil v červnu 2015 v seznamu pěti set nejvýkonnějších superpočítačů na světě na čtyřicáté místo [35].

## 2.1 Paralelní výpočet

V paralelních systémech můžeme rozlišit tři základní paralelní programovací modely, a to sdílený adresový prostor, datově paralelní model a zasílání zpráv, které nás bude zajímat nejvíce [6]. V případě sdíleného adresového prostoru mluvíme o vláknech, které rozlišují lokální proměnné na zásobníku a sdílené proměnné na hromadě, skrze které navzájem komunikují [14]. Datově paralelní model mapuje funkce na kolekce dat. Díky mnohonásobným výpočetním jednotkám se každý prvek kolekce zpracovává nezávisle bez potřeby komunikace s ostatními jednotkami. Posledním z modelů je zasílání zpráv mezi procesy. Každý proces tak má vlastní kopii dat, kterou může synchronizovat s ostatními procesy zasíláním zpráv skrze síťové rozhraní.

Jestliže je nějaký kód paralelizován, je to s cílem urychlit výpočet, který sekvenčně probíhá příliš dlouho. Paralelizovaný kód je pak obvykle podroben řadě testů, ze kterých jsou určeny jeho základní vlastnosti. Nejdůležitější je pro nás zrychlení, tedy kolikrát je paralelní čas kratší než čas sekvenčního běhu. Pokud jsme vypočetli zrychlení, můžeme vypočíst také účinnost, a to tak, že hodnotu zrychlení vydělíme počtem výpočetních jednotek.

Dalším velmi důležitým parametrem popisujícím vlastnosti algoritmu je škálování [10], a to především protože současný vývoj vysoce výkonných výpočtů jde cestou zvyšování výpočetního výkonu rostoucím počtem procesorů zapojených do jediného clusteru [12].

Rozlišujeme silné škálování, které, jak Ahmdal poukázal, neumožňuje neomezeně zvyšovat počet výpočetních jednotek a předpokládat, že se stejnou měrou bude zvětšovat zrychlení. To není možné, protože některé části programu nelze paralelizovat jako jiné a navíc samotná paralelizace s sebou přináší také určitou míru režie. Pokud by sekvenční část tvořila deset procent výpočtu, je teoreticky možné paralelizací zcela zredukovat zbylou část výpočtu, ale sekvenční část nezměníme. Z toho vyplývá, že maximální možné zrychlení tohoto případu je desetkrát.

Oproti Ahmdalovu zákonu postavil Gustafson slabé škálování, kdy s narůstajícím počtem výpočetních jednotek roste také velikost řešeného problému, čímž většinou relativně klesá sekvenční část úlohy. Díky této skutečnosti je možné řešit větší úlohu ve stejném čase jako úlohu menší.[1]

### 2.1.1 Message Passing Interface

Message Passing Interface (MPI) [17] je specifikací rozhraní pro zasílání zpráv mezi procesy. Tato specifikace také obsahuje podporu pro dynamickou tvorbu procesů nebo paralelní vstupně-výstupní operace. Ač se nejedná o standardizovanou specifikaci, přesto je MPI široce používáno a podporují jej všechny superpočítačová centra. Velká obliba tohoto rozhraní je spojena s její efektivitou, spolehlivostí a přenositelností, což jsou hlavní cíle MPI.

MPI meziprocesová komunikace je založena na komunikátorech. Všechny procesy v komunikátoru se vyznačují rankem, který umožňuje jejich identifikaci. Pokud používáme některou z MPI funkcí, vždy je jedním z parametrů komunikátor, který nám umožní identifikovat a propojit požadované procesy. Po inicializaci MPI rozhraní v kódu paralelizované aplikace má uživatel k dispozici komunikátor MPI_COMM_WORLD, s kterým může pracovat, nebo z něj vytvořit jiné komunikátory.

[1] Dr David Henty z Edinburgh Parallel Computing Centre přirovnal Ahmdalův a Gustafsonův zákon k letu letadlem. Jestliže letíme do Londýna, zajisté tam doletíme novodobým concordem rychleji než běžným letem, avšak odbavení na letišti nemůžeme urychlit, jedná se tedy o sekvenční část letu. Pokud bychom však místo do Londýna letěli do Washingtonu, výrazně zredukujeme čas potřebný na odbavení vůči celkovému času letu, a tím také dosáhneme lepšího zrychlení.

V MPI rozlišujeme dva typy komunikací, a to point-to-point a kolektivní. V případě poit-to-point funkcí, probíhá komunikace jen mezi dvěma procesy, jeden je odesilatel, druhý příjemce zprávy. Příjemce a odesilatele rozpoznáme podle jejich ranku v komunikátoru, nad kterým je funkce volána.

Naopak kolektivní operace vyžadují, aby byly zavolány všemi procesy v komunikátoru. Existuje zde mnoho funkcí na přeposílání dat, lišících se v tom, které procesy se stanou odesilateli nebo příjemci informace (v kolektivních operacích, které mají za příjemce všehcny procesy, se stává odesilatel zárovň příjemcem) a které odesílané hodnoty příjemci dostanou (všechny, část nebo jedinou hodnotu z odeslaných dat). Vedle zmíněných funkcí existují další kolektivní funkce jako například operace redukce, synchronizační bariera nebo operace vytváření nových komunikátorů.

Jelikož kolektivní operace se vyznačují tím, že danou funkci musí zavolat všechny procesy v komunikátoru, vytváříme nové komunikátory především s cílem vyloučit část procesů z těchto operací. Nový komunikátor můžeme vytvořit rozštěpením některého ze stávajících komunikátorů, s informací o ranku procesu v novém komunikátoru a takzvanou barvou procesu, která proces zařadí do jedné ze skupin komunikátoru. Nadále kolektivní operace nad daným komunikátorem musí provést všechny procesy spadající pod stejnou barvu.

Dalším důležitým znakem MPI funkcí je míra synchronizace, kterou funkce vyžadují. Dělíme proto funkce na blokující a neblokující. V případě blokujících operací dochází k synchronizaci všech procesů volajících danou funkci. Žádný z procesů tak nemůže pokračovat v provádění výpočtu, dokud funkci nezavolají všechny procesy v komunikátoru, nad kterým je funkce prováděna. Oproti tomu procesy volající neblokující funkce nečekají na zbylé procesy, ale pokračují ve výpočtu dále, avšak dříve než proces přistoupí k cílovému/zdrojovému, který byl pro danou neblokující operaci použit, musí provést kontrolu, zda požadovaná funkce již skončila. Většina MPI funkcí existuje v blokující i neblokující variantě, tudíž je na uživateli, kterou z nich zvolí.

### MPI-I/O

Součástí MPI specifikace je také MPI-I/O [16], tedy část rozhraní určená pro paralelní vstupně-výstupní operace.

Pro paralelní I/O je možné využít POSIX operace, kdy je pomocí seek nastavena pozice v souboru dle požadovaného počtu bajtů od začátku souboru. Výhodou MPI-I/O je, že poskytují podporu pro kolektivní i nezávislé paralelní operace, neblokující I/O nebo přístup do několika nesouvislých částí souboru najednou.

Rozdíl v kolektivních a nezávislých paralelních I/O operacích je v kooperaci procesů při přístupu do souboru. Nezávislý přístup definuje pouze oblasti souboru, kam daný proce přistupuje, který při této operaci nijak nespolupracuje s ostatními procesy. Opakem jsou kolektivní I/O operace, které využívají synchronizovaného zápisu pro optimalizaci přístupu k úložišti požadovaného souboru. Od vydání poslední MPI specifikace 3.1 jsou podporovány také neblokující kolektivní I/O operace [17].

Operace MPI-IO mají však nevýhodu, že stejně jako POSIX, pracují výhradně s binární soubory, což znesnadňuje práci s nimi. Na tomto rozhraní je však postavena řada vysokoúrovňových I/O knihoven, které zjednodušují práci se soubory a poskytují řadu flexibilnějších formátů výstupních souborů. Příkladem takovéto knihovny může být HDF5, které je věnována sekce 2.2.3.

### 2.1.2 Allinea Distributed Debugging Tool

Vývoj softwarových produktů je složitá činnost, kterou je vhodné usnadnit skrze nástroje, které pomohou lépe porozumět chování vyvíjeného programu. Nejčastější je potřeba ladících nástrojů (debuggerů), pomocí kterých je prováděna dynamická analýza programu s cílem lokalizovat chyby v kódu [28]. Jelikož většina běžných nástrojů nepodporuje vývoj paralelních aplikací, je nutné využít programů vytvořených pro tyto účely. Jednou ze společností zabývajících se touto problematikou je Allinea, jejíž nástroj Distributed Debugging Tool (DDT) [1] je dostupný na superpočítači Salomon.

Allinea DDT je programem pro ladění paralelních aplikací, který umožňuje analyzovat aplikace běžící až na tisíci procesorech. Vedle klasického sledování vývoje stavu programu nabízí sledování paměti s detekcí neuvolňování dynamicky alokované paměti nebo čtení a zápis mimo hranice polí.

V místech označených bodem přerušení (breakpointem) vkládá DDT MPI bariéru, která zastaví v daném místě provádění procesů a umožní nahlédnout do stavu paměti každého jednoho procesu v místě bariery. Samozřejmá je také možnost postupného krokování výpočtu, v tomto případě daný krok provedou vždy všechny procesy.

Pro možnosti ladění je nutný překlad s parametrem -g a doporučen je také parametr -O0, pomocí nihž jsou vypnuty optimalizace překladače, aby ladící nástroje mohly jednoznačně spojit prováděný kód se zdrojovým textem [7].

### 2.1.3 Allinea MAP

Dalším důležitým nástrojem je Allinea MAP [2], který slouží na profiling, což je další z důležitých postupů při vývoji paralelních aplikací. Především u superpočítačových programů je snaha minimalizovat celkový čas běhu a zajistit co nejlepší škálování. Právě škálovatelnost je jedním z nejdůležitějších rysů superpočítačových algoritmů. S rostoucím počtem dostupných procesorů je důležité vytvářet aplikace, které je možné provádět na více jádrech s co nejmenší ztrátou účinnosti [26] [10]. Proto je využíváno profilingových nástrojů, které ukazují, v kterých částech kódu strávily procesy (případně vlákna) jaké množství času a z jakých důvodů. Umožňují tak nalezení úzkých míst implementace a oblasti pro potenciální optimalizaci.

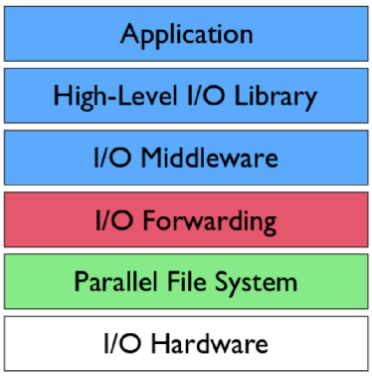
Nástroj MAP ukazuje vývoj spotřeby operační paměti, množství operací v pohyblivé řádové čárce, a také jakou činností tráví jednotlivé procesy čas. Všechny tyto hodnoty jsou zaznamenávány po celou dobu běhu aplikace a poté jsou pro uživatele vykresleny do samostatných grafů ukazujících jejich vývoj.

MAP rozlišuje mezi čtyřmi aktivitami výpočetních procesů, které rozlišuje v odpovídajícím grafu různými barvami. Zelená část grafu indikuje čas strávený samotným výpočtem. Modrou barvou jsou zaznamenány MPI komunikace a čekání procesů při provádění blokujících MPI funkcích. Další část grafu je oranžová, jež indikuje čas na vstupně-výstupní operace. A poslední detekovanou veličinou je čas strávený čekáním procesorů na dokončení činnosti akcelerátorů a vrácení výpočtu zpět. Tato část je tmavě fialová.

V hlavním okně nástroje MAP, který ukazuje naměřené výsledky, je zobrazen kód analyzované aplikace, kde jsou označeny řádky se signifikantní časovou spotřebou spolu s danou mírou obarvenou dle odpovídajícího důvodu.

## 2.2 I/O subsystém na superpočítačích

Vstupně výstupní systém se skládá z několika vrstev, přičemž každá z nich zásadně ovlivňuje výkonnost paralelního I/O. Tyto vrstvy jsou vyobrazeny na Obrázku 2.1.




Obrázek 2.1: Vrstvy I/O subsystému na superpočítačích [15].

- I/O hardware
  Vstupně výstupní hardware je na superpočítačích tvořen poli pevných disků. Některé superpočítače poskytují také datová úložiště tvořená magnetickými páskami, určená pro zálohu uživatelských dat, sporadicky se také vyskytují SSD disky.

- Paralelní souborový systém
  Paralelní souborový systém spravuje datový hardware. Navazuje na předchozí vrstvu skrze síťové přepínače, případně specializované I/O storage servery. Oproti běžným souborovým systémům poskytují paralelní systémy podporu pro vysoký výkon při sdílení vícero uživateli a umožňují škálovatelné vstupně-výstupní operace pomocí rozložení zátěže přes dostupné úložiště. Typickým příkladem takovéhoto systému je pNFS [23], nebo Lustre [21], kterému je věnována sekce 2.2.2.

- I/O Forwarding
  I/O forwarding směruje vstupně výstupní požadavky na dedikované I/O uzly, které je přesměrovávají na paralelní souborový systém. Požadavky jsou zde přeskládány a následně agregovány s cílem snížit latenci systému [20]. Tato vrstva je však přítomná jen na některých z největších superpočítačových systémech a je typicky úzce spojená s hardwarem [20][15].

- I/O middleware
  V případě využívání Message Passing Interface (MPI) a paralelního I/O je na této úrovni MPI-I/O standart. Stejně jako MPI standart, také MPI-I/O má mnoho implementací, nejrozšířenější z nich je ROMIO [4].

- Vysokoúrovňová I/O knihovna
  Pro paralelní zápis je možné využívat některou z implementací MPI-I/O, avšak nad touto vrstvou jsou postaveny ještě vysoko-úrovňové I/O knihovny, které kromě zápisu pomocí MPI-I/O přináší navíc také správu metadat, optimalizace pro zápis do překrývajících se datových oblastí nebo současný zápis a čtení různými procesy a mnoho dalších. Příklady takovýchto knihoven jsou netCDF [37] nebo HDF5 [32].

Je nezbytné mít na mysli, že každá z uvedených vrstev má mnoho nastavení a uživatel by měl, dříve než začne optimalizovat svůj kód, najít vhodné nastavení předchozích vrstev.

### 2.2.1 I/O subsystém na superpočítači Salomon

Systém ukládání dat na Salomonu [27] tvoří dvě datová úložiště HOME a SCRATCH, která poskytují 454 TiB, respektive 1 538 TiB. HOME je určen pro skladování dat, která uživatelé na superpočítači dlouhodobě potřebují, proto je HOME zálohován na kapacitní diskové pole a také na páskovou knihovnu. Uživatelské úlohy by však měly zapisovat do SCRATCH, který je pro to určen a poskytuje výrazně vyšší rychlost zápisu a čtení. Na SCRATCH je instalován souborový systém Lustre, který se skládá ze dvou metadata serverů a šesti object storage serverů, které spravují 54 object storage targetů. Význam těchto částí je vysvětlen v sekci 2.2.2, která je věnována souborovému systému Lustre. Nástroj k-Wave využívá pro zápis knihovnu HDF5, která ukládá data do vlastního souborového formátu určeného pro velká vědecká data. Více o HDF5 je uvedeno v sekci 2.2.3.

### 2.2.2 Souborový systém Lustre

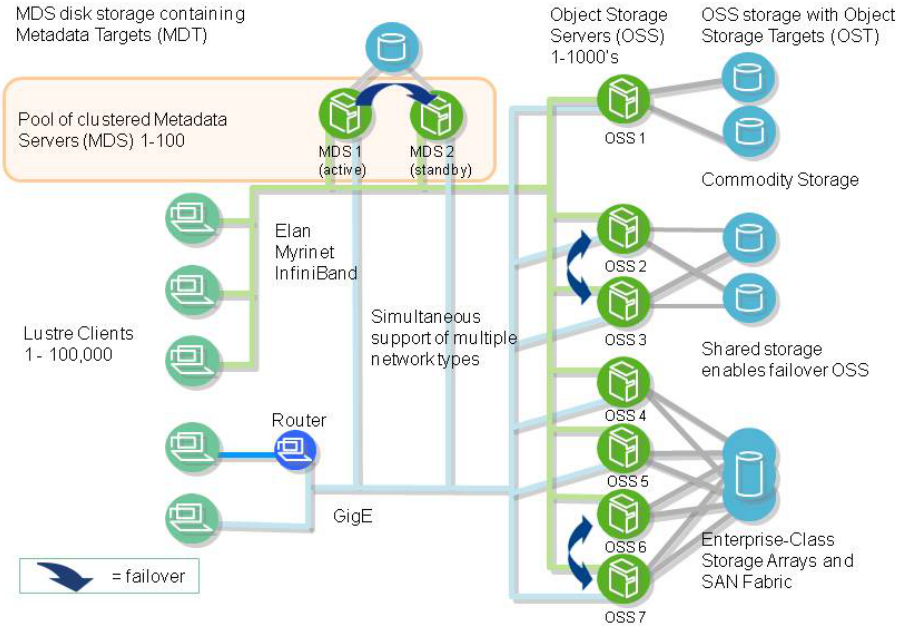
Lustre [21] je široce používaný paralelní souborový systém. Tento souborový sytém využívá většina z předních superpočítačů, a to především, protože poskytuje přístup až desetitisíci uživatelům současně, umožňuje uložit mnoho data včetně velkých petabajtových souborů, ale také poskytuje vysokou rychlost I/O operací [25].

Lustre odděluje ukládání metadat a samotných souborů pro lepší škálovatelnost a spolehlivou opravu souborů při selhání díky kombinaci výhod žurnálování a distribuovaného souborového systému [5].

Lustre se skládá z několika částí 2.2 [22]:

- Metadata Servers (MDS)
  Metadata Server spravuje uložení metadat na některý z Metadata Target. Uchovávají se zde také záznamy změn metadat pro případ obnovení dat po vzniku selhání.
- Metadata Target (MDT)
  Na Metadata Target jsou uložena metadata, takže zde najdeme názvy souborů a adresářů, jejich přístupová práva nebo rozložení souborů.
- Object Storage Servers (OSS)
  Podobně jako metadata, ani data nejsou uložena přímo, ale skrze Object Storage Server, který zpracovává datové požadavky souborů, které jsou uloženy na některém z Object Storage Target, které spravuje. Pod jeden OSS spadají dva až osm OST.
- Object Storage Target (OST)
  Na Object Storage Target jsou datové objekty z jednoho nebo více OSS.
- Lustre Client
  Lustre Client je uzel s nainstalovaným příslušným Lustre software, který vytváří rozhraní mezi Lustre servery a linuxovým virtuálním souborovým systémem.

Typickým optimalizačním způsobem paralelních souborových systémů je rozložení zátěže přes dostupné pevné disky. V případě Lustre se jedná o nastavení počtu a velikosti Lustre stripe. Tato technika ukládá data na požadovaný počet OST tak, že zapisovaná data jsou rozdělena na bloky o velikosti jednoho stripe, přičemž jednotlivé bloky jsou cyklicky rozloženy mezi vybrané OST.




Obrázek 2.2: Ilustrace propojení komponent souborového systému Lustre [22].

### 2.2.3 Hierachical Data Format

The Hierarchical Data Format (HDF) [32] je open source souborový formát podporující vytváření velkých, komplexních, heterogenních souborů. HDF formát byl vyvinut ve Spojených státech amerických v Národním centru pro superpočítačové aplikace (National Center for Supercomputing Applications, NCSA) pro snazší uložení a manipulaci s vědeckými daty napříč různými operačními systémy a stroji [18].

Tato knihovna je široce využívaná pro práci s daty na superpočítačích. Jedním z uživatelů je například Národní úřad pro letectví a kosmonautiku (National Aeronautics and Space Administration, NASA), který se také podílel na vývoji HDF verze 5 (HDF5), která podporuje paralelní zápis a čtení do/z jediného souboru.

Významnými rysy HDF5 souborů jsou:

- Možnost paralelního zápisu.
- Do HDF5 souborů je možné uložit libovolné datové typy.
  Příkladem mohou být čísla, obrázky nebo videa.
- Vytvořené soubory jsou nezávislé na architektuře počítače a jsou tak plně přenositelné.
- Soubory jsou samopopisné.
  Metadata popisující daný soubor jsou uloženy odděleně od zbytku dat.

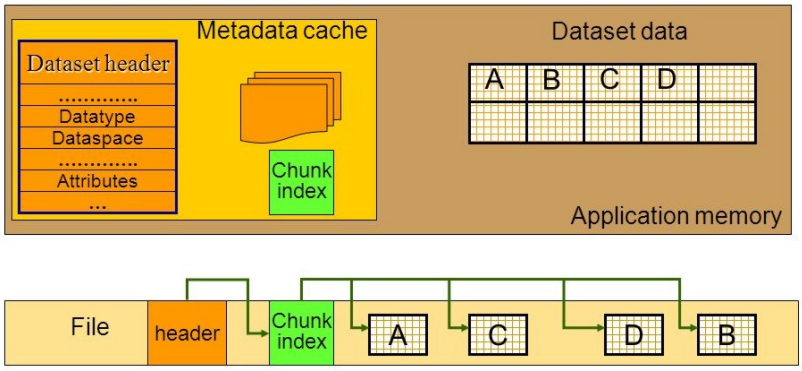
Další z výhod HDF5 formátu je, že je podporován všemi nejrozšířenějšími vizualizačními nástroji, což umožňuje snadnější analýzu dat.

HDF5 soubor [31] je kontejnerem pro HDF5 objekty, přičemž každý soubor je tvořen minimálně jedním objektem, a to kořenovou skupinou. Skupina je v HDF5 obdobou adresáře, v níž jsou umístěny jednotlivé objekty, kterými jsou další skupiny nebo datasety. Dataset je multidimenzionálním polem datových elementů, jehož velikost a dimenzionalita se popisuje pomocí dataspace objektů. Dataspace a datový typ datasetu jsou dány v době

jeho vytváření a později není možné je pro daný dataset změnit. Stejně tak je tomu u properties (vlastností), které udávají další nastavení jako je komprese datasetu nebo jeho typ. Pokud na počátku neznáme celkovou velikost datasetu, musíe vytvořit rozšiřitelný dataset, kterému před samotným zápisem do něj uvedeme o kolik se dataset zvětší.

V HDF5 jsou tři typy datasetů a to:

- Souvislý
  Datové pole datasetu je bráno jako jeden velký celek, který je tak také uložen na disku.
- Chunkovaný
  Dataset je rozdělen na malé části pevné velikosti, které nazýváme chunky. Veškeré vstupně-výstupní operace pracují s bloky zaokrouhlenými na jednotlivé chunky. Hlavním důvodem k zavedení chunkovaného datasetu je, že v případě velkých souborů na disku, a především v operační paměti nemusí být tak velký souvislý blok paměti. Na Obrázku 2.3 je naznačeno jak chunkovaný soubor vypadá.
- Kompaktní
  Kompaktní typ datasetu je určen výhradně pro malé soubory (menší než 64kB). Takovýto dataset je uložen spolu s metadaty, což zajistí vyšší rychlost práce s daty.




Obrázek 2.3: Vyobrazení uložení chunkovaného datasetu [13].

Při vytváření objektů můžeme modifikovat jejich chování pomocí takzvaných property listů, a umožnit tak například paralelní práci s datasetem.

Pro jednotlivé datové objekty je možné definovat atributy, které jsou uloženy v metadatech daného souboru a popisují daný objekt. Díky atributům, které jsou pevně spjaty s jednotlivými objekty, získáváme podrobný popis obsahu souboru, aniž bychom museli vytvářet další soubory, které by tuto informaci obsahovaly. Říkáme proto, že HDF5 soubory jsou samopopisné.

Pro paralelní práci s HDF5 soubory je klíčový hyperslab, pomocí kterého může každý proces vybrat pouze určitou oblast v datasetu, se kterou chce pracovat. Podobně jako při vytváření datasetu musíme definovat dimenzionalitu hyperslabu, což uděláme opět pomocí dataspace, a jeho posunutí oproti souřadnicovému začátku datasetu, nad kterým je hyperslab vytvořen. Zápis nebo čtení pak probíhá do, respektive z takto vytvořeného hyperslabu.

Knihovna HDF5 kromě rozhraní pro zápis a čtení dat do souborů v HDF5 formátu obsahuje také mnoho nástrojů pro práci s HDF5 soubory [33]. Příkladem těchto nástrojů

jsou h5ls pro vypsání metadat daného souboru (uživatel tak získá informaci o velikosti a dimenzionalitě jednotlivých datasetů včetně popisků), h5dump pro výpis obsahu souboru nebo h5diff, pomocí kterého je možné porovnat obsah dvou souborů, a mnoho dalších nástrojů, které zjednodušují práci s těmito soubory.

### Postup zápisu a čtení HDF5 souboru

Paralelní zápis v HDF5 proto začínáme vytvořením property listu pro modifikaci přístupu k souboru. Všechny procesy, které budou s tímto souborem pracovat, musí být v jediném komunikátoru, který je vstupním parametrem funkce H5Pset_fapl_mpio(). Následně s tímto property listem vytváříme soubor voláním H5Fcreate(). Dále musíme vytvořit další property list, abychom skrze H5Pset_dxpl_mpio() nastavili, zda chceme provádět kolektivní nebo nezávislý zápis do souboru.

Jestliže známe celkovou velikost datasetu, pokračujeme vytvořením dataspace, kde parametrem H5Screate_simple() je pole s rozměry jednotlivých dimenzí. V opačném případě mluvíme o rozšiřitelném datasetu, kterému definujeme velikost dimenzí stejným způsobem, avšak s neomezenou hodnotou.

Protože obvykle vyžadujeme chunkovaný dataset (chunkování je pro rozšiřitelné datasety nutné), musíme vytvořit další property list, tentokrát pro tvorbu datasetu. Pro tento property list definujeme velikosti dimenzí chunků funkcí H5Pset_chunk().

Nyní konečně můžeme vytvořit dataset, do kterého budeme zapisovat data. To provedeme pomocí funkce H5Dcreate2(), které kromě uvedených parametrů musíme uvést datový typ a umístění datsetu v souboru.

Jak již bylo uvedeno dříve, pro paralelní přístup k souboru musíme vytvořit hyperslab. Pro hyperslab je nutné definovat jeho velikost vytvořením dalšího dataspace. Poté musíme získat přístup k cílovému datasetu skrze H5Dget_space(). V tomto datesetu vytvoříme hypeslab pomocí H5Sselect_hyperslab(), pro který kromě vytvořeného dataspace musíme nastavit posunutí vůči počátku datasetu odpovídajícím $N$-dimenzionálním polem.

Nyní již každý z procesů může zapsat do souboru H5Dwrite(), a to skrze svůj hyperslab. Pokud bychom však pracovali s rozšiřitelným datasetem, musíme ještě před samotným zápisem zavolat funkci H5Dextend(), a provést tak rozšíření datasetu na aktuální požadovanou velikost.

Paralelní čtení HDF5 souboru probíhá podobnou posloupností kroků, kdy otevřeme dataset a vytvoříme v něm hyperslab. Z oblasti dané hyperslabem pak může každý proces číst. V tomto případě ale již pro soubor nic nenastavujeme, naopak musíme na počátku zjistit velikost datasetu, který chceme číst. K zjištění velikosti datasetu požadovaného jména slouží funkce H5LTget_dataset_info().

V průběhu čtení i zápisu dochází k vytvoření velkého množství HDF5 objektů, jedná se o property listy, datasety, dataspace a také o samotný soubor. Poté, co tyto objekty již nepotřebujeme, je nutné všechny uzavřít.

# Kapitola 3

# Zápis dat

Paralelní zápis můžeme popsat jako transfer dat, které jsou rozdistribuovány v pamětech jednotlivých procesů, do souboru (případně více souborů), který může být rozložen přes několik pevných disků. Obecně se jedná o problém mapování rozložení dat z pamětí výpočetních uzlů na rozložení na disku. Toto mapování může být značně komplikované, proto je optimalizace mapování kritická pro výkon paralelního I/O [19].

V simulačních nástrojích dochází k ukládání vypočtených dat každou $N$-tou iteraci pro post-processing a vizualizaci, zároveň však tyto aplikace často musí podporovat ukládání veškerých dat výpočtu pro vytvoření záchytného bodu, díky kterému je možné výpočet pozastavit a následně z tohoto souboru obnovit.

Jsou zde tři základní vzory, jak ukládat data v paralelních aplikacích [11].

- 1-1
  Data jsou sesbírána do jediného procesu, který je všechna zapíše do souboru. Tento způsob zápisu je sériový, a tedy velmi jednoduchý, avšak tato technika neškáluje a je možné ji použít pouze pro malý počet procesů s malým množstvím dat. Sesbírání dat od ostatních procesů je časově náročné a může být nemožné z důvodu nedostatku paměti.

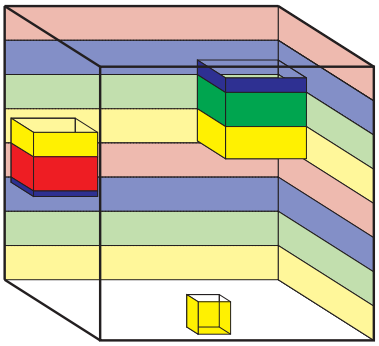
- N-N
  V tomto případě aplikace s $N$ procesy vytváří $N$ nezávislých souborů – jeden pro každý proces. Zápis do těchto souborů také nepřináší implementační komplikace, přesto tato technika skrývá řadu problémů. Aplikace generuje velké množství malých souborů během jediného běhu, které je těžké a zároveň nutné následně zpracovat.

- N-1
  Každý z procesů zapisuje svá data do jediného sdíleného souboru. *N-1* způsob zápisu nemá problémy předchozích, nicméně tento vzor ukládání dat vykazuje velmi nízkou propustnost při zápisu malých datových celků.

V paralelním I/O rozlišujeme dva typy přístupů k zápisu a to individuální a kolektivní. Při individuálním zápisu zapisuje každý proces svá data nezávisle na ostatních (bez komunikace nebo koordinace s jinými procesy), což je vhodné především pro zápis velkého množství dat. Kolektivního zápisu se musí účastnit všechny procesy, což teoreticky umožňuje optimalizaci zápisu především malých požadavků.

## 3.1 Požadavky na zápis v k-Wave

V k-Wave se pracuje s tří-dimenzionálním prostorem, v kterém se vypočítává průběh šíření ultrazvuku. Tato doména je distribuována mezi jednotlivé procesy rozdělením jedné z dimenzí tak, jak je to znázorněno na Obrázku 3.1. Jelikož není nutné ukládat veškerá data, definujeme uvnitř domény jednu nebo více subdomén, které obalují oblasti, které nás zajímají. Reálným příkladem může být simulace šíření ultrazvuku ze zářiče na tumor na játrech pacienta. Na Obrázku 3.2 je výpočetní doména, uvnitř níž jsou dvě subdomény, které obalují zářič a tumor.



Obrázek 3.1: Subdomény uvnitř domény, která je rozdělena mezi procesy po deskách.

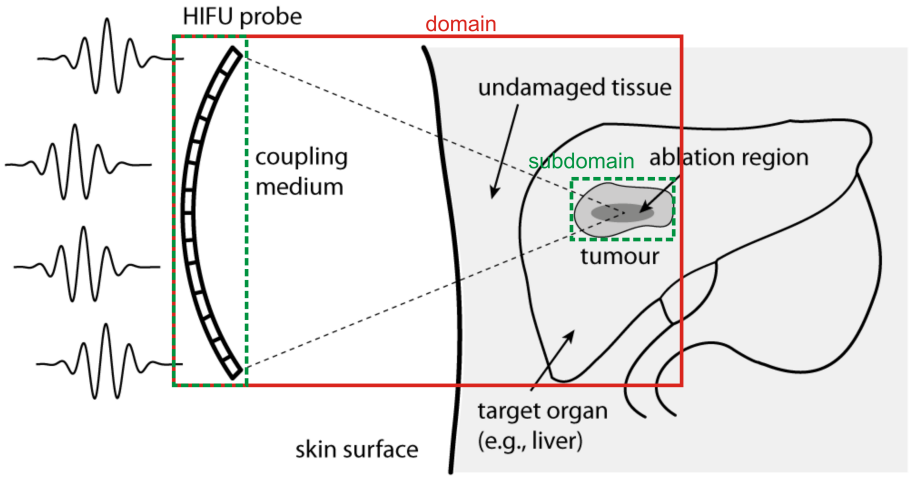
V každém kroku simulace se pak ukládají nové hodnoty uvnitř subdomén, což z hlediska zápisu znamená, že rozložení dat na zápis mezi procesy není rovnoměrné. Některé procesy tak nezapisují vůbec, jiné procesy naopak mohou spravovat několik subdomén a zapisovat tak několikanásobně více dat než jiné. Problémem jsou především některé tvary subdomén, přidělující malé množství dat na proces na zápis. Tyto malé zápisy vykazují vysokou režii, která způsobuje velmi pomalý zápis. Rozdíl v rychlostech paralelního zápisu různého množství dat byl změřena a je popsána v sekci Testy 4.

Jelikož je kolektivní zápis blokující operací, není možné pokračovat ve výpočtu, dokud nezapíše data nejpomalejší z procesů. A protože pro analýzu výsledků simulace je nutné uložit data z velkého množství iterací, stává se zápis dat významnou časově náročnou operací, která vyžaduje optimalizaci.

## 3.2 Optimalizace

Základním předpokladem optimalizace zápisu dat jsou vhodně vyladěné jednotlivé vrstvy cluster data storage. Avšak pokud aplikace bude zapisovat nevhodně rozložená data, pak ani není v možnostech jednotlivých vrstev dosáhnout maximální rychlosti zápisu. Cílem proto je najít způsob, jak ukládat data rychleji z pohledu rozložení dat mezi procesy.

Víme, že požadavky na zápis malého množství dat jsou zdrojem pomalého zápisu, jelikož způsobují příliš režie, která rychlost zápisu zcela degraduje. Abychom si tento fakt ukázali v reálných číslech, provedl jsem test srovnávající rychlost zápisu rozdílného množství dat na jednotlivé zapisující procesy, který jsem opakoval s různým počtem procesů. Ty jsou voleny jako násobek velikosti výpočetního uzlu na superpočítači Salomon, což je dvacet čtyři,




Obrázek 3.2: Ultrazvukový vysílač zářící na tumor na játrech pacienta s vyobrazením výpočetní domény a dvou subdomén [9].

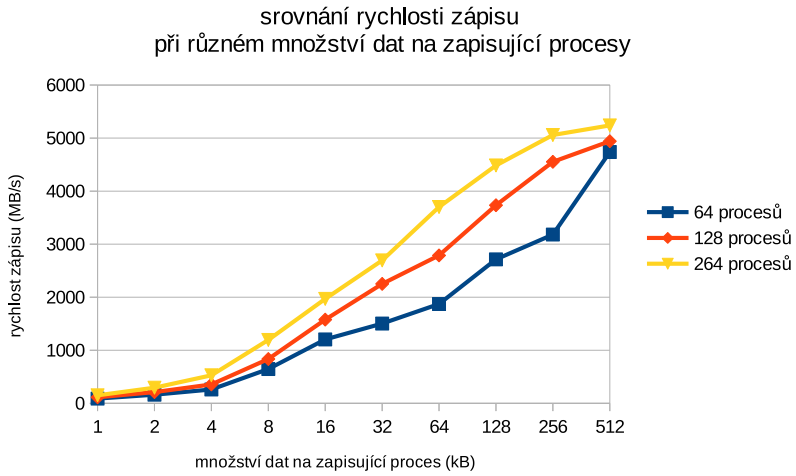
nebo jako mocniny dvou jelikož takovýto počet procesů používáme při simulaci v k-Wave, abychom rozdělili simulační doménu, jejíž velikost bývá také nastavena na mocninu dvou, beze zbytku. Jak zde vidíme, rychlost zápisu kilobajtů se pohybuje na úrovni stovek megabajtů za sekundu a hranice jednoho gigabajtu za sekundu je překročena nejdříve při zápise osmi kilobajtů a to pouze 264 procesy. S rostoucím množstvím dat rychlost velmi rychle roste a překvapivě má tento parametr výrazně větší vliv na rychlost než celkový počet procesů.

V případě mnoha malých požadavků na zápis, existují zde způsoby, jak zápisy optimalizovat – dvoufázový zápis a sesypání dat, které jsou implementovány v ROMIO. Dvoufázový zápis (two-pahase I/O) [30] je technika, která se využívá, pokud jsou data procesů při kolektivním zápise proložená (interleaved data). V první fázi probíhá komunikace mezi procesy, kdy zjistí, zda a jakým způsobem se data prokládají. Každý proces vytvoří dočasný buffer o stejné velikosti jako jsou jeho data a následně procesy provedou výměnu dat tak, aby každý proces měl celistvý kus dat. Druhou fází je již zápis dat z dočasného bufferu do souboru. Dvoufázový zápis se tak využívá jen v případě kolektivního zápisu, protože procesy při něm spolupracují. Druhou technikou je sesypání dat (data sieving) [30] pro optimalizaci I/O v rámci každého procesu samostatně, kdy nespojitá data určená pro zápis se nejprve nahrají do pomocného bufferu již bez dat, které nechceme zapsat. Takto se spojí několik malých zápisových požadavků v jeden, který je následně zapsán větší rychlostí. Pro tuto techniku je limitující velikost pomocného bufferu, který je pro zápis 512kB a pro čtení 4MB, jeho velikost je však možné změnit pomocí "MPI-IO's hints mechanism" [30].

Uvedené techniky zajišťují výrazné zrychlení zápisu, avšak pokud je požadavků na zápis jen pár, není téměř možné je jakkoli optimalizovat. V tom případě musíme pro zápis zvolit jiné řešení. Nabízí se několik variant, které jsou rozebrány dále.

### 3.2.1 Delegované uzly nebo procesy

V případě delegovaných uzlů/procesů a stejně tak neblokujícího zápisu mluvíme o technice write-behind, kdy zápis probíhá zároveň s výpočtem hodnot následující iterace. Ideálně by měl zápis být rychlejší než výpočet nových hodnot, aby se doba čekání výpočetních procesů minimalizovala.




Graf 3.3: Test srovnávající rychlost zápisu při změně množství dat na zapisující proces.

Volba delegovaných procesů může být vhodná například na strojích, které mají procesory o atypickém počtu jader, jako například superpočítač SuperMUC Phase 2, který je osazen procesory Haswell Xeon Processor E5-2697v3, které mají 14 jader. Výpočetní doménu je přinejmenším nevhodné dělit na takovýto počet částí, zvláště pokud se jedná o prvočísla, proto se využití jednoho nebo dvou procesů pro delegovaný zápis nabízí.

Nevýhodou je potřeba napsat kód tak, že se data budou odesílat na vybrané delegované procesy. Výběr procesů delegovaných na zápis by pravděpodobně musel zůstat na uživateli, který určí nejvhodnější variantu pro daný stroj. Zároveň je tímto způsobem část výpočetního výkonu věnována výhradně na I/O, což při některých simulacích nemusí být výhodné, především v případě, kdy data pro zápis má jen málo procesů z celkového množství procesů provádějících výpočet.

### 3.2.2 Neblokující zápis

Další variantou je neblokující kolektivní zápis, který funguje podobně jako delegované procesy, ale v tomto případě je zásah do kódu minimální. V tomto případě je zápis proveden na pozadí a procesy po zavolání odpovídajících funkcí okamžitě pokračují ve výpočtu a nečekají na dokončení dané operace. Zásadním problémem tohoto řešení je, že se jedná o novinku a v HDF5 ještě není implementována.

### 3.2.3 Transformace rozložení dat mezi procesy

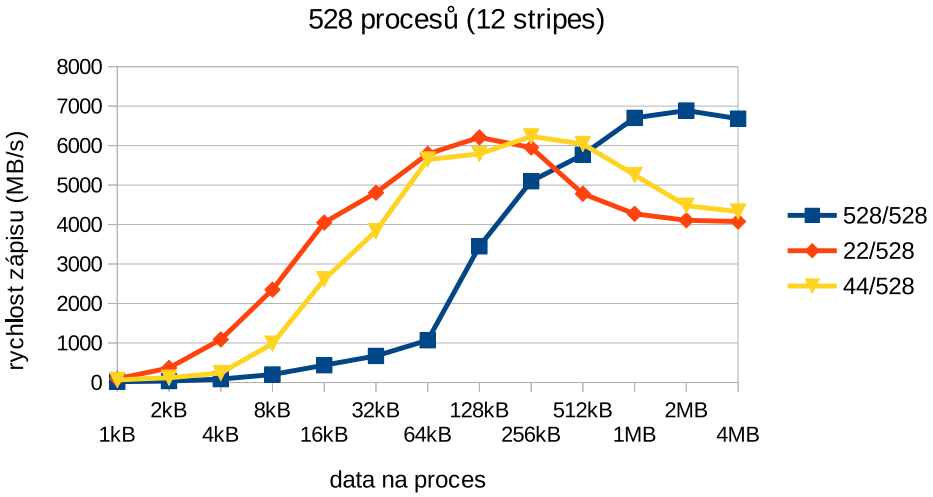
Třetí možností je provést transformaci rozložení dat, kdy využíváme volání MPI funkcí MPI_Send() spolu s MPI_Recv() nebo MPI_Gatherv() k přeposílání dat mezi MPI procesy. Cílem takovéhoto přeposílání je v první řadě vyrovnání množství dat na zapisující procesy. Jelikož kolektivní paralelní zápis je blokující operací, nemůže se pokračovat ve výpočtu, dokud zápis neukončí poslední zapisující proces, což především v případě nerovnoměrného množství zapisovaných dat může znamenat časté čekání.

Realizace obecného algoritmu, který zajistí rovnoměrné rozložení dat, by spočívala v přeposílání určité části většiny procesů, což by se stalo úzkým hrdlem takovéto implementace.

Zároveň je vhodné přeposílat data jen mezi sousedními procesy (v opačném případě by procesy zapisovaly najednou do několika oblastí v souboru, čímž dochází k vyšší složitosti nastavování zápisu a především k štěpení zápisových požadavků, o čemž je napsáno v sekci 3.2.4 věnující se kumulaci dat), a to nás v možnostech přeposílání dat do jisté míry omezuje.

Z pohledu uživatele je proto možným způsobem vybalancování jednotlivých procesů jejich rozdělením do skupin, kde rozdíl sum dat procesů v každé skupině je minimální. Následně je možné nastavit přeposílání dat do jediného procesu z této skupiny, díky čemuž dosáhneme požadovaného efektu.

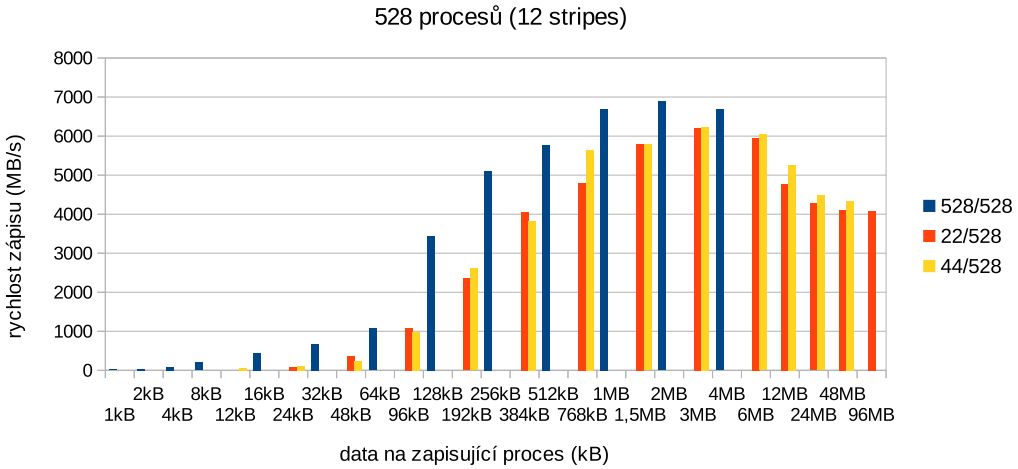
Další možnost optimalizace založené na přeposílání dat byla uveřejněna v práci [38] zabývající se optimalizací a parametrů vstupně-výstupního rozhraní a teoretickou možností redukovat počet zapisujících procesů a agregovat tak množství zapisovaných. Tato varianta optimalizace měla pro zápisy v řádu kB velmi dobré výsledky, proto jsem zopakoval tento test.




Graf 3.4: Srovnání rychlosti různých typů zápisu při různém množství na dat na každý z 528 procesů.

Na Grafu 3.4 jsou vyobrazeny výsledky testu dosažené rychlosti zápisu při využití 528 procesů, které zapisovaly do souboru s dvanácti Lustre stripes. Výsledkem je zápis, který v tomto případě dosáhla hranice 7 GB/s. Jak je z grafu vidět, při zápisu malého množství dat na zapisující proces rychlost zápisu nedosahuje ani zlomku maximální rychlosti. Proto, stejně jako v uvedené práci [38], byl tento zápis všech procesů srovnán se zápisem, kdy jsou data přeposlána na jediný proces na uzel (každý dvacátý čtvrtý proces), respektive na socket (každý dvanáctý proces), který následně tato data zapsal.

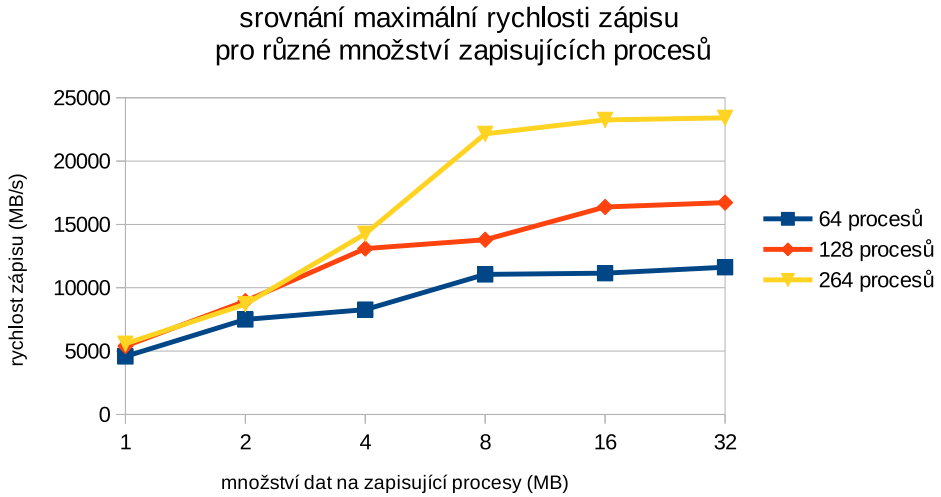
Výsledkem tohoto testu tak je potvrzení, že množství dat na zapisující proces je významnějším parametrem než počet zapisujících procesů, jak již bylo uvedeno dříve u Grafu 3.3, a tedy i tento způsob kumulace dat, který vyžaduje přidané přeposílání dat, přinesl signifikantní zvýšení rychlosti (největšího zrychlení bylo dosaženo při 4 kB dat na proces, který při zápisu všemi procesy dosáhl rychlosti 84 MB/s, zatímco při kumulaci dat na uzel rychlost zápisu přesáhla 1 GB/s, což znamená zrychlení téměř 13krát), a to až do úrovně stovek kilobajtů dat na proces. Za touto hranicí již byla rychlost zápisu všemi procesy vysoká a zároveň stoupla doba přeposílání dat, tudíž klesla také rychlost zápisu s předchozí kumulací.




Graf 3.5: Srovnání rychlosti různých typů zápisu při daném množství dat na zapisující proces.

Na následujícím Grafu 3.5 jsou naměřené hodnoty z předchozího experimentu posunuty tak, abychom porovnali, jaké rychlosti zápisu dosahuje zapisující proces, který má dané množství dat. Ve všech případech můžeme sledovat, že maximální rychlosti zápisu je dosaženo při zápisu od jednoho MB na zapisující proces. Toto množství tedy můžeme považovat jako ideální pro dosažení maximální rychlosti (tento závěr je samozřejmě možné vztáhnout jen na daný způsob zápisu a testovaný superpočítač).

V obou uvedených případech snižujeme celkový počet zapisujících procesů, musíme však mít na mysli, že tím zároveň také snižujeme maximální možnou dosaženou rychlost. V Grafu 3.6 můžeme vidět, jaké rychlosti bychom mohli dosáhnout, pokud bychom zapisovali dostatečně velké množství dat, kde je evidentní, že tento strop je dán počtem procesů.




Graf 3.6: Srovnání vlivu počtu zapisujících procesů na maximální možnou rychlost zápisu.

### 3.2.4 Kumulace dat

Kumulace dat přes několik iterací s odloženým zápisem do chvíle naplnění bufferu je jednoduchým způsobem zvýšení množství dat na zapisující procesy a zdá se být ideální volbou k řešení zápisu malého množství dat. Tímto způsobem bychom mohli sbírat data z celého běhu aplikace a zapsat až na závěr, což však není vhodné řešení a to nejen z důvodu omezené operační paměti. Proto je při kumulaci zásadní otázkou přes kolik iteraci data kumulovat a dle toho vytvořit potřebné buffery.

Původním předpokladem bylo určit, při jakém množství dat dosáhneme s daným počtem procesů maximální rychlosti, stejně tak jako jsme to určili z Grafu 3.5, a poté kumulovat data přes jednotlivé iterace, dokud nedosáhneme odpovídajícího množství. Pro určení ideálního množství iterací jsem navrhl I/O analytickou aplikaci, pomocí které je možné provést analýzu cílového stroje. Touto aplikací jsem tak pro daný počet procesů určil, jaké dosáhneme rychlosti, pokud budeme zapisovat určité množství dat. Tímto způsobem došlo ke zjištění, jaké množství dat je pro každý konkrétní případ ideální a dle toho parametru jednotlivé procesy nastavily počet iterací přes které data kumulovat, aby dosáhly tohoto množství dat.

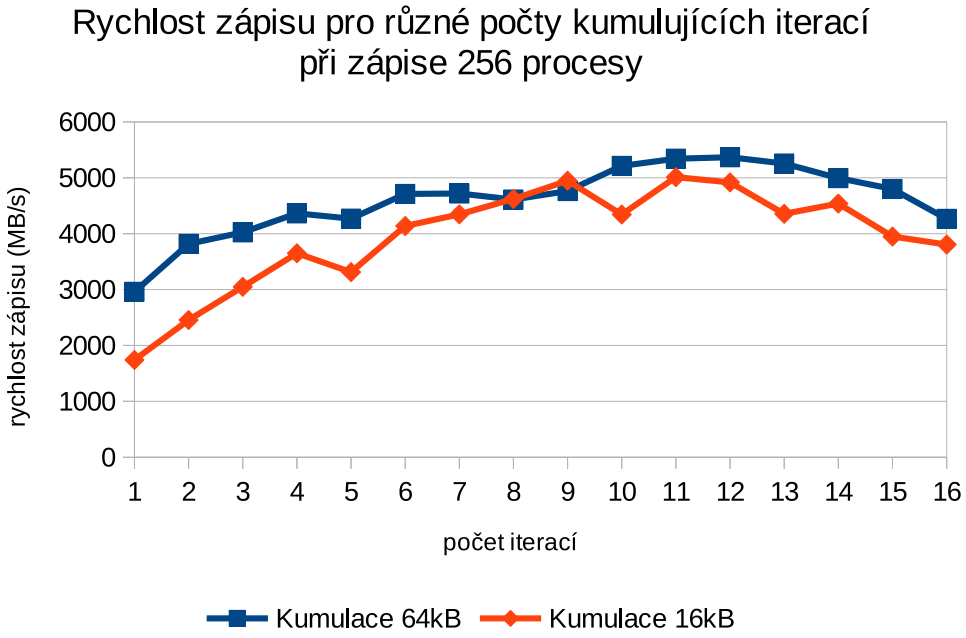
Jelikož zápis musí provádět všechny procesy ve stejnou iteraci, je nutné všechny procesy synchronizovat, aby kumulovaly stejný počet iterací. Proto byl výstup I/O analytické aplikace doplněn o časy, jak dlouho jednotlivé zápisy trvají. Na základě těchto časů byl vybrán takový počet iterací, který zajišťoval nejvyšší rychlost zápisu.

Tyto předpoklady se však ukázaly jako mylné, jelikož nebylo vzato v úvahu, že zápis je zásadně ovlivněn dimenzionalitou dat, přičemž kumulací v čase rozšiřujeme druhou dimenzi zapisovaných dat. Vysoká dimenzionalita dat je častým problémem aplikací, které zapisují dostatečně velké množství dat, ale jejich paměťová oblast je rozdělena podle nejvyšší dimenze. Takto na místo jediného požadavku na velké množství dat vzniká mnoho menších požadavků, což je opět zdrojem pomalého I/O (vstupně-výstupní operace nedosahují maximální možné rychlosti).

Tento problém je v knihovně HDF5 částečně řešitelný, a to pomocí rozšíření dimenzí HDF5 chunků, které jsou nejmenší zapisovatelnou/čtenou jednotkou při vstupně-výstupních operacích s HDF5 soubory. Rozměry chunku proto byly nastaveny tak, že kopírují zapisovaná data. S takto nastaveným chunkem byl provecen test srovnávající vývoj rychlosti při kumulování různého množství dat po dobu jedné až šestnácti iterací. Výsledky tohoto testu jsou zaneseny v Grafu 3.7.

Jak je z tohoto grafu vidět, rychlost zápisu spolu s narůstajícím počtem iterací, přes které probíhá kumulace dat z počátku výrazně roste, a to dokonce rychleji než rychlost zápisu daného množství dat. Rozdíl v zápise stejného množství dat v jednotlivých případech se liší pouze dimenzionalitou dat a HDF5 chunků, z čehož lze usoudit, že data umístěná pouze v jediné dimenzi nejsou po zápis nejvýhodnější. Srovnání takovýchto zápisů je zaznamenáno také v Tabulce 3.1.

Z dat v Grafu 3.7 především vyplývá, že není možné tímto způsobem provádět kumulaci dat neomezeně dlouho. Po určité době již rychlost zápisu neroste a je vhodné kumulování dat ukončit. Jelikož vhodný počet iterací pro kumulaci se pro jednotlivé množství dat různí (pro Graf 3.7 byly vybrány hodnoty s podobnými výsledky pro lepší srovnání), což je dobře vidět například v ukázkovém výstupním souboru I/O analýzy 3.1, byla I/O analytická aplikace upravena tak, aby určovala počet iterací, s kterými zápis dosáhne nejvyšší rychlosti.




Graf 3.7: Graf srovnávající rychlost zápisu při kumulaci dat přes různé množství iterací.

| | Kumulace 64kB | | Kumulace 16kB | |
|---|---|---|---|---|
| Data (kB) | rychlost zápisu (MB/s) | iterace | rychlost zápisu (MB/s) | iterace |
| **64** | 2 960,77 | 1 | 3 649,60 | 4 |
| **128** | 3 817,67 | 2 | 4 620,39 | 8 |
| **192** | 4 024,44 | 3 | 4 917,00 | 12 |
| **256** | 4 363,88 | 4 | 3 805,30 | 16 |

Tabulka 3.1: Srovnání rychlosti zápisu stejného množství dat nakumulovaného v různém množství iterací.

## 3.3 Metoda optimalizace paralelního zápisu

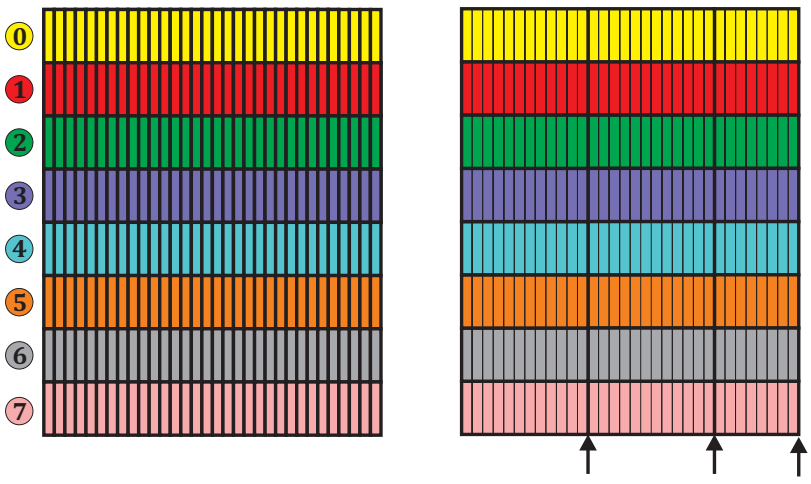
Na základě uvedených možností optimalizace jsem navrhl nový způsob zápisu, kombinující jednotlivé postupy. Optimální způsob zápisu byl v průběhu vývoje průběžně modifikován, čímž vzniklo několik variant. Všechny metody byly okamžitě implementovány do k-Wave, na které byly také testovány. Jak již bylo deklarováno dříve, navržený způsob zápisu je možné využít v libovolné aplikaci, která cyklicky generuje data na zápis rozložené mezi procesy. Avšak vývoj určený pro k-Wave tuto metodu omezuje v jednom ohledu, a to, že veškeré parametry musí být nastaveny na počátku simulace, jelikož v průběhu simulace jsou zakázány alokace paměti jako ochrana proti neočekávanému pádu aplikace. Není tak možné za běhu zvětšovat nebo vytvářet pomocné buffery. Abychom byli schopni na počátku simulace rozhodnout, jaké parametry zápisu jsou nejvýhodnější, vytvořil jsem aplikaci, která provede I/O analýzu daného výpočetního clusteru, která nám poskytne veškeré důležité informace, na základě kterých nastavíme vhodné hodnoty parametrů. Této aplikaci je věnována následující sekce 3.5.

V původní verzi k-Wave probíhal zápis nativně, tudíž tak, že všechny procesy po vypočtení hodnot z nové iterace simulace přesunuly data daná sensorovou maskou do bufferu pro zápis. Sensorová maska je vektor indexů pozic uvnitř domény, které označují, která výstupní data se mají uložit do souboru. Po překopírování takto označených dat byl na závěr iterace obsah tohoto bufferu zapsán do souboru.

### 3.3.1 Kumulace dat

Prvním způsob optimalizace je založen výhradně na kumulaci dat s cílem nasbírat v jednotlivých procesech minimální množství dat, s kterým je možné dosáhnout maximální možné rychlosti zápisu. Při tomto způsobu se navíc využívá přeposílání dat k eliminaci rozdílů množství dat mezi procesy. Za sebou následující procesy s malým množstvím dat (rychlost zápisu daného množství dat vykazuje v porovnání s rychlostí procesu s největším množstvím dat menší rychlost než je zadaný parametr) se proto shlukují do skupin. V takto vytvořených skupinách přebírá vždy jediný proces data od ostatních ve skupině.

Pro každý proces s daty pro zápis (po přeuspořádání) je určeno ideální množství kumulujících iterací. Tento způsob optimalizace poté synchronizuje procesy na základě výpočtu rychlosti zápisu z doby potřebné na zápis odpovídajícího množství dat. Z tohoto důvodu výstup I/O analýzy obsahuje časový údaj jednotlivých zápisů. Počet iterací, přes které probíhá kumulace, je omezen množstvím operační paměti, kterou uživatel parametrem na tuto optimalizaci přidělil Toto omezení je nutné, jelikož lze předpokládat velký počet kumulujících iterací, které by tak mohly v některých procesech nashromáždit příliš velké množství dat. Jak se liší nativní zápis s touto metodou zápisu je vyobrazeno na Obrázku 3.8. Při nativním zápise každý proces zapisuje každou iteraci jeden malý obdélník, zatímco tato metoda provede zápis až po nakumulování dostatku dat, což je naznačeno šipkami pod touto reprezentací souboru.




Obrázek 3.8: Srovnání oblastí zápisu jednotlivých procesů nativním zápisem a zápisem po kumulaci dat přes několik iterací v případě rovnoměrného rozložení dat na zápis.

Problémem tohoto řešení je takřka neomezené množství kumulujících iterací. S odloženým zápisem roste dimenze času v zapisovaných datech, což musí být reflektováno také rozšířením dimenze HDF5 chunku. Jelikož vstupně-výstupní operace jsou rozděleny na chunky, bez zvětšení chunků by došlo k rozdělení daného zápisu zpět na původní velikost. Chunky by však neměly v dimenzi času příliš růst, jelikož by vznikl problém při na-

čítání dat. Namísto načtení stavu subdomény v jediné iteraci by došlo ke čtení hodnot ze všech iterací, které chunk pokrývá, což by především v případě velkého chunku mohlo vést k významným časovým problémům.

### 3.3.2 Kumulace dat s redukcí počtu zapisujících procesů

Jelikož slabinou předchozího způsobu optimalizace je velký počet kumulujících iterací, bylo nutné jej vhodně omezit. Aby došlo k většímu nakupení dat na zápis, je navíc zavedena agregace dat z okolních procesů. V případě nerovnoměrného rozložení dat dochází také k přeuspořádání dat s cílem minimalizovat tyto rozdíly.

Na počátku běhu aplikace proto nejprve dojde k rozdělení procesů do skupin. Skupiny tvoří procesy se stejným množstvím dat na zápis, které následují ihned za sebou. Následně dochází ke spojování skupin o jednom procesu se skupinami, s kterými sousedí, za předpokladu, že taková skupina existuje a že na procesy v dané skupině připadá více dat než na proces, který zůstal samotný. Cílem je sjednotit původně zamýšlené subdomény v jednu skupinu, kde tyto samostatné procesy byly na kraji domény a proto na ně připadlo méně dat na zápis.

Každá ze skupin je reprezentována množstvím zapisovaných dat na proces (množství připojených okrajových procesů nebereme v úvahu) a počtem zapisujících procesů z dané skupiny. V tuto chvíli jsou všechny skupiny označeny počtem zapisujících procesů, který odpovídá celkovému počtu procesů ve skupině. V dalších krocích však budeme tento počet snižovat.

Dále dojde k vyhledání skupiny, v níž na jednotlivé procesy připadá největší množství dat. Ve snaze minimalizovat rozdíly v množství dat na zapisující procesy je ve zbylých skupinách určen počet procesů z dané skupiny, které by měly sbírat data od okolních procesů ze skupiny pomocí funkce MPI_Gatherv() a získat tak podobné množství jako procesy s nejvíce daty.

Pokud ve všech skupinách stále připadá na zapisující procesy malé množství dat (do stovek kilobajtů) a je zároveň možné ve všech skupinách zredukovat počet zapisujících procesů na polovinu až osminu. Jestli je možné takto zredukovat počet zapisujících procesů je dáno tím, zda je možné takto zredukovat počet zapisujících procesů ve skupině s nejmenším počtem zapisujících procesů.

Na základě takto nastaveného počtu zapisujících procesů jsou v každé skupině procesy označeny barvou nového MPI komunikátoru pro přeuspořádávání dat, kde každá skupina je rozdělena do menších skupin označením vždy jinou barvou komunikátoru tak, že každý zapisující proces má vlastní barvu a spolu s ním do této barvy patří následující procesy, od kterých bude tento proces data přebírat. V případě, že skupina byla tvořena pouze zapisujícími procesy, nedochází samozřejmě pro tyto procesy k vytváření nové barvy v tomto komunikátoru, ale jsou zařazeny spolu s procesy bez dat na zápis do nulté barvy, čímž se předchází zbytečnému přeposílání dat.

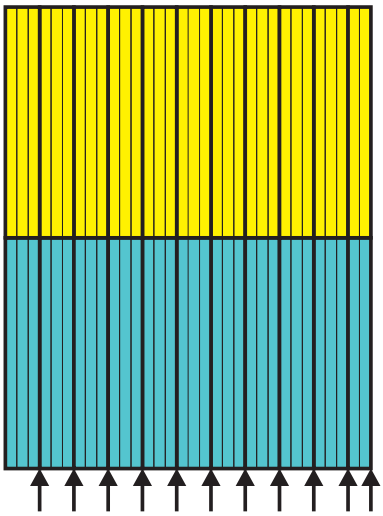
K jednotlivým barvám jsou dopočteny také ranky a je vytvořen nový MPI komunikátor, pomocí kterého jsou nastaveny parametry funkce MPI_Gatherv(), které využíváme v každé iteraci pro přeposlání dat prvnímu procesu (rank 0) v dané barvě komunikátoru. Tímto způsobem je, tak zajištěna optimalizace transformací rozložení dat mezi procesy.

Dale je vytvořen také nový MPI komunikátor pro zapisující procesy, aby došlo k vyloučení zbylých procesů ze zápisu. Všechny zapisující procesy poté určí ideální počet iterací, přes které data kumulovat a to na základě vstupního souboru s I/O analýzou daného počítače. Proto v případě spuštění simulace bez I/O analýzy nejsou prováděny žádné opti-

malizace zápisu, jelikož není možné určit, přes kolik iterací je vhodné provádět kumulaci. Každý zapisující proces si zjistí hodnotu z této analýzy pro dané množství zapisovaných dat. Protože v analýze jsou jen výsledky měření pro určité hodnoty, je obvykle nutné aproximovat hodnotu na základě dvou okolních hodnot. V této implementaci byla jako aproximační metoda zvolen průměr.

Ač přeuspořádání dat bylo iniciováno snahou minimalizovat rozdíly, nemuselo se to povést (kupříkladu v situacích, kdy rozdíly jsou enormní nebo počet procesů ve skupinách malý), tudíž jednotlivé zapisující procesy mohly určit jiný ideální počet kumulujících iterací. Aby byl zápis proveden ve stejné iteraci, musí být určena jediná hodnota. Jelikož již malý počet kumulujících iterací má výrazný rychlostní přínos a naopak větší množství iterací by mohlo mít opačný efekt (viz sekce 3.2.4 o kumulaci dat), byl zvolen výběr minima z hodnot všech procesů.

Podle hodnoty kumulujících iterací jsou alokovány buffery, do kterých se v každé iteraci přesouvají data zapisujích procesů, respektive jsou do nich přeposlána data z procesů od kterých daný proces data sbírá, a následně po naplnění bufferu (končí-li simulace v iteraci, která není násobkem kumulovaných iterací, tak dříve) jsou data zapsána do výstupního HDF5 souboru.



Obrázek 3.9: Oblasti zápisu procesů do souboru při způsobu zápisu s redukcí počtu zapisujících procesů.

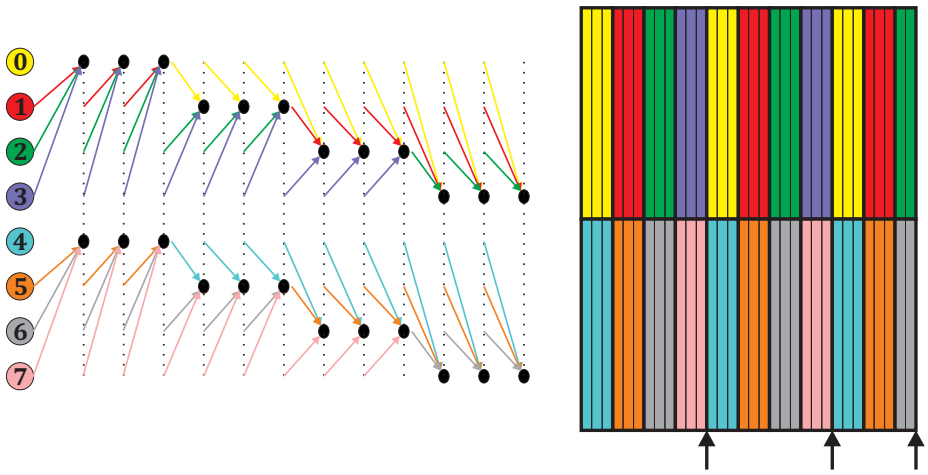
Tento způsob zápisu je naznačen na Obrázku 3.9. Při srovnání s předchozím zápisem z Obrázku 3.8 dochází k zápisu mnohem častěji (každou třetí iteraci), ale zápis provádí jen dva procesy z osmi, které převzaly data od následujících procesů, které s nimi tvořili skupinu.

Jelikož je tento způsob zápisu založen na kumulaci dat přes jednotlivé iterace, mělo by stejně jako v předchozím případě dojít ke zvětšení chunku v časové dimenzi. V tomto případě však obvykle nedochází ke kumulaci takového velkého množství iterací jako při metodě samotné kumulace, proto jsem nastavil velikost této dimenze chunku stejnou jako počet kumulujících iterací. Nicméně v situacích, kdy dojde k zápisu po nakumulování většího množství iterací nebo celkové množství dat při jednom zápise naroste, je vhodné chunk rozdělit na několik menších částí.

### 3.3.3 Změna rozložení dat

Jelikož předchozí metoda snižovala celkový počet zapisujících procesů, docházelo tím zároveň ke snížení potenciální hranice maximální možné rychlosti zápisu tak, jak to bylo prezentováno v sekci 3.2.3. Aby k tomu nedocházelo, byla předchozí metoda upravena tak, že v případě agregace dat redukující počet zapisujících procesů ve všech skupinách, do kterých jsou procesy rozděleny, nedochází k zápisu po nakumulování daného počtu iteraci. Naopak se poté začnou agregovat data stejným způsobem do následujícího procesu ve skupině, který také nakumuluje dané množství dat. Takto postupně opět využijeme všech možných procesů, jedná se o kombinaci předchozích přístupů, kdy dochází ke změně rozložení zapisovaných dat.

Tento způsob zápisu je naznačen na Obrázku 3.10, kde v první části vidíme posloupnost přeposílání dat mezi procesy a v druhé části oblasti, kam procesy nově zapisují.




Obrázek 3.10: Názorná ukázka přesunu dat mezi procesy a soubor s označenými oblastni zápisu jednotlivých procesů ve třech zápisech metodou změny rozložení zapisovaných dat.

### 3.3.4 Označení subdomén kuboidy

Předchozí uvedené metody optimalizace zápisu jsou navrženy pro případ označení subdomén pomocí senzorové masky. Tento způsob označení subdomén umožňuje nastavit její libovolný tvar, ale vede k pomalému následnému zpracování, jelikož je nutné data načítat po jednotlivých indexech daných maskou.

Pro zlepšení rychlosti zpracování výstupních souborů je proto nutné nahradit senzorovou masku takzvanými kuboidy, subdoménami ve tvaru kvádru obalujícími oblasti zájmu. Toto řešení označení dat pro zápis vede obecně k většímu množství ukládaných dat, protože nejsme schopni definovat libovolný tvar subdomény, ale umožní při post-processingu načítání dat po blocích, což vede k výrazně rychlejšímu načítání dat. Možnost označování subdomén rohy kuboidů je implementována v několika jiných verzích k-Wave, ale v MPI verzi dosud chyběla.

Zápis pomocí kuboidů přináší do problematiky několik změn. V případě využití senzorové masky jsou data z každé iterace v souboru v jediném lineárním poli, zatímco kuboidy jsou v rámci výstupního souboru zapsány do samostatného datasetu. Tyto datasety jsou čtyřdimenzionální, přičemž první doménou je čas, zbylé tři domény tvoří daný kuboid.

Tento způsob rozdělení kuboidů do samostatných datasetů však způsobuje z hlediska paralelního zápisu problém. Jelikož paralelní zápis je nastavován na základě MPI komunikátoru na celý soubor, je nutné a, by zápis do všech datasetů byl prováděn všemi procesy z daného komunikátoru. Protože soubor můžeme mít otevřen pouze jednou s jediným komunikátorem, musíme zápis jednotlivých kuboidů provádět sekvenčně.

Z hlediska optimalizace zápisu v tomto případě nemusíme vyhledávat skupiny procesů, jelikož ty jsou nyní dány hranicemi kuboidů. Zároveň je nutné vzít v potaz, že zápis vícero kuboidů nemůže probíhat současně, proto není nutné synchronizovat zápis všech subdomén na shodné iterace. Naopak je vhodné nastavit tento parametr tak, aby na závěr jednoho kroku simulace proběhl zápis co nejmenšího počtu subdomén, aby nedocházelo ke vzájemnému blokování. Ve zbylých ohledech je možné využít předchozích metod optimalizace zápisu. Zavedením kuboidů tedy získáme rychlejší načítání dat při post-processingu, ale zápis samotný bude v případě několika subdomén pomalejší.

V rámci této práce byla z časových důvodů implementována pouze optimalizace kumulací, avšak z technického hlediska není problém implementovat pro kuboidy také nejsložitější metodu změny rozložení dat.

## 3.4 Implementace v k-Wave

Nově vytvořené verze k-Wave vyžadují spouštění s dvěma novými parametry. Pomocí parametru -w je zadána cesta k XML souboru s I/O analýzou a parametr -f slouží u optimalizace kumulací dat k nastavení do dostupné volné paměti.

V *k-SpaceFirstOrderSolveru* je před vytvářením výstupních streamů zavoláno natavení I/O parametrů. Jedná se o novou funkci ve třídě *Parameters*, která jednotlivým procesům nastaví základní parametry zápisu, množství zapisovaných dat a pozici do které části souboru bude daný proces zapisovat. Jestliže byla simulace spuštěna s I/O analýzou, dojde k načtení obsahu daného XML souboru. K tomu dochází v nově vytvořené třídě *IOParameters*, ve které pro zpracování XML je využita běžná knihovna TinyXML-2 [34].

Následně se ještě z třídy *Parameters* zavolá funkce *SetIOParameters()* ve třídě *IOParameters*, která nastaví zbylé vstupně výstupních parametry. Nejprve tak dojde k nastavení parametrů pro přeposílání dat mezi procesy dle dané metody optimalizace ve funkci *SetIOGatherv()* [1], poté dojde ve funkci *SetIOWrite()* k vytvoření nového MPI komunikátoru, který obsahuje jen procesy zapisující do souboru. Posledním nezbytným krokem nastavování vstupně výstupních parametrů je určení ideálního počtu kumulujících iterací.

Jestliže jsou všechny I/O parametry nastaveny, je možné vytvořit požadované výstupní streamy. V každém streamu se při vytváření nově nastaví parametry pro přeposílání dat mezi procesy a alokuje pro tyto účely pomocný buffer. Také se požadovaným způsobem zvětší buffer pro zápis, aby se do něj vešla data ze všech kumulujících iterací.

Výpočet v k-Wave dále probíhá běžným způsobem až do chvíle, kdy je pro daný stream zavolána funkce pro zápis dat *WriteDataDirectly()*. Procesy přeposílající data je odešlou pomocí MPI_Gatherv() a zbylé procesy je překopírují do zápisového bufferu. Pokud je zápisový buffer naplněn nebo probíhá poslední iterace simulace, dojde k zápisu obsahu zápisového bufferu do souboru.

Pro umožnění zápisu subdomén daných kuboidy bylo nutné provést další změny ve zdrojovém kódu k-Wave. Prvně muselo dojít k úpravě načítání vstupních dat ve třídách *Ma-*

[1] V metodě optimalizace kumulací s agregací procesů a metodě změny rozložení je funkce *SetIOGatherv()* nahrazena funkcí *SetIOGatherv2()*.

*trixContainer* a *LongMatrices*. Poté se ve třídě *IOParameters* v nové funkci *CreateCuboids()* vytváří vektor obsahující záznamy struktury *TCuboidInfo* s veškerými důležitými informacemi o jednotlivých subdoménách. Dále třída *OutputScatteredRealstream* je doplněna o funkcí *CreateCuboidDatasets()*, která vytváří požadované čtyř-dimenzionální datasety pro každý kuboid. Aby to bylo možné, bylo nutné upravit také třídu pro práci s HDF5 soubory a doplnit ji o funkce podporující práci s 4D datasety.

## 3.5 I/O analýza

Uvedené metody optimalizace I/O v k-Wave nově nastavují ideální počet iterací pro kumulování dat na základě vstupního souboru s analýzou rychlostí zápisu provedené aplikací vytvořenou pro tyto účely. Tento způsob optimalizace je vynucen zákazem dynamických alokací před začátkem samotné simulace. Veškerá nastavení tak probíhají na počátku běhu aplikace a později jej není možné měnit (kumulace dat z menšího množství iterací je možná, ale není žádaná).

Aplikace provádí zápis způsobem odpovídajícím zápisu v k-Wave a měří dobu zápisu s cílem zaznamenat časy pro jednotlivá nastavení. Tato aplikace tak provádí zápis všemi procesy s uniformním rozložení dat, kdy na začátku testování připadá na každý z procesů množství dat dané vstupním parametrem, poté se v každém následujícím kroku toto množství zdvojnásobí, dokud nedosáhneme požadované horní hranice.

Pro optimalizaci samotnou kumulací dat byl zaznamenáván čas samotného zápisu daného množství dat. S omezením počtu kumulujících iterací je informace o čase doplněna počtem iterací. Aby aplikace správně určila tuto hodnotu, je pro každé množství dat měřen také přínos kumulace přes několik iterací. Proto analýza začíná zápisem jediné iterace, poté počet iterací zvyšuje, avšak jen do doby dokud kumulace vede ke zvýšení rychlosti zápisu. Jelikož naměřené rychlosti mohou mírně oscilovat, dochází k ukončení zvětšování kumulací, když několik za sebou jdoucích měření nevedlo k zvětšení dosažené rychlosti.

Pro všechna nastavení je zápis opakován (počet iterací dle zadaného parametru), přičemž vždy je změřen čas jednotlivých iterací. Měření probíhá pomocí funkce MPI_Wtime(), kterou každý z procesů změří dobu trvání zápisu, následně je provedena redukce a je vybrána největší hodnota. Z takto naměřených hodnot je odstraněn první čas, který je ovlivněn tvorbou výstupního souboru, a ze zbylých hodnot je vybrán medián, který je zanesen do výsledného souboru.

Vstupní parametry testovací aplikace:

-d = pokud je parametr zadán, smaže vytvořené HDF5 soubory
-f = název výsledného souboru (bez přípony .xml), pokud není zadán, výsledek je vypsán na stdout
-i = do výstupního souboru je přidána informace, kterou uživatel zadal v argumentu
-r = počet iterací zápisu pro každé množství dat na zápis
-w = počátek názvu vytvořených HDF5 souborů (bez přípony .h5)
název souboru je vždy doplněn o zapisované množství dat na proces
-n = minimální velikost zapisovaných dat na proces v kB
-x = maximální velikost zapisovaných dat na proces v kB

Výsledky analýzy jsou zapsány ve formátu XML, což je široce rozšířený a pro člověka čitelný typ souboru [2]. Soubor generovaný touto aplikací je tvořen informacemi o testu, které

[2] XML formát výsledků I/O analýzy přináší možnost snadné modifikace hodnot, což umožňuje s nastavením I/O dále experimentovat.

jsou určeny uživateli pro snadnější správu jednotlivých souborů. Uživatel se tak zde dočte, kdy daný test proběhl, kolik procesů test provádělo, v jakých jednotkách jsou naměřené hodnoty, počet iterací, ze kterých je test vyhodnocen, a případně další informace, které uživatel může přidat pomocí parametru testovací aplikace.

V následující části souboru jsou uvedeny výsledky daného testu. Každý záznam je tvořen informací o množství zapisovaných dat na jeden proces, čas, jak dlouho zápis probíhal, rychlost zápisu a ideální počet kumulací pro dosažení maximální rychlosti při daném množství dat. Pro aplikaci načítající tuto analýzu jsou směrodatné informace o množství dat, čase a počtu iterací, přičemž zbylé informace jsou zde uvedeny pro uživatele pro lepší čitelnost naměřených výsledků. Čistě pro uživatele je zde také uvedena informace, za jakých parametrů v průběhu celého průběhu analýzy bylo dosaženo úplně nejvyšší rychlosti zápisu.

Příkladem výstupního souboru jedné z analýz je uvedený soubor 3.1, který vznikl jako výsledek analýzy rychlosti zápisu 264 procesů (jedenáct uzlů o dvou procesorech s dvanácti jádry) na superpočítači Salomon.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- k-Wave server IO test -->
<test>
  <testInfo>
    <version>2</version>
    <date>2016-4-21</date>
    <procs>264</procs>
    <units>
      <data>kB</data>
      <speed>MB/s</speed>
      <time>s</time>
    </units>
    <iterations>100</iterations>
    <info>it4i Salomon</info>
  </testInfo>
  <results>
    <record data="1" speed="149.791" time="0.00172114" iterations="12" />
    <record data="2" speed="293.604" time="0.00175619" iterations="9" />
    <record data="4" speed="527.807" time="0.00195384" iterations="9" />
    <record data="8" speed="1194.2" time="0.0017271" iterations="9" />
    <record data="16" speed="1968.09" time="0.00209594" iterations="11" />
    <record data="32" speed="2693.47" time="0.00306296" iterations="11" />
    <record data="64" speed="3701.05" time="0.00445819" iterations="14" />
    <record data="128" speed="4484.29" time="0.00735903" iterations="25" />
    <record data="256" speed="5058.64" time="0.013047" iterations="24" />
    <record data="512" speed="5239.16" time="0.0251949" iterations="15" />
  </results>
  <bestSpeed data="256" iterations="24" speed="5648.25" />
</test>
```

Soubor 3.1: Výstup I/O analýzy pro počítač Salomon pro 264 procesů.

Testovací aplikace by měla být na cílovém počítači vždy spouštěna se stejným počtem procesů, jaký bude následně zapisovat při běhu optimalizované aplikace, jelikož pro různé

množství procesů se ideální množství iterací pro zápis liší. Zároveň pro každý způsob uložení dat v souboru by měla existovat odpovídající varianta této aplikace, provádějící zápis stejným způsobem. Takto vytvořený výstup analýzy je však možné použít pro všechny běhy aplikace s odpovídajícím počtem zapisujících procesů.

Pro zápis dat touto aplikací, stejně jako samotným optimalizovaným programem, nesmíme zapomenout, že je zásadní provádět zápis do souborů v adresářích na scratch oddílech, které mají nastaveny lustre stripes zajišťující distribuci zapisovaných dat přes jednotlivé Object Storage Targety.

# Kapitola 4

# Testy

Následující testy byly provedeny na superpočítači Salomon v Ostravě a zkoumají možné případy subdomén uvnitř domény o velikosti 512 x 512 x 512 bodů mřížky. Každá ze simulací probíhala po dobu tisíce iterací, ve kterých byla generována data pro zápis. Volba podoby subdomén vychází z běžných požadavků aplikace k-Wave, přičemž se jedná především o případy vykazující nízkou rychlost zápisu. Na obrázcích zobrazující jednotlivé testované případy jsou hrany jedné z dimenzí domény vždy vyznačeny přerušovanou čarou, což značí, že právě tato dimenze je rozdělena mezi procesy provádějící výpočet.

Ve všech uvedených případech byla změřena doba zápisu všech procesů, z niž byl vybrán nejdelší z časů. Z takto naměřených časů všech zápisů byl odstraněn první záznam, který obvykle trvá déle než zbylé zápisy, protože v HDF5 dochází k vytváření potřebných struktur, a poslední záznam, jelikož poslední zápis mohl být tvořen menším počtem zapisovaných iterací. Tyto časy jsou následně seřazeny a je z nich vybrán medián, který je přepočten na výslednou rychlost zápisu.

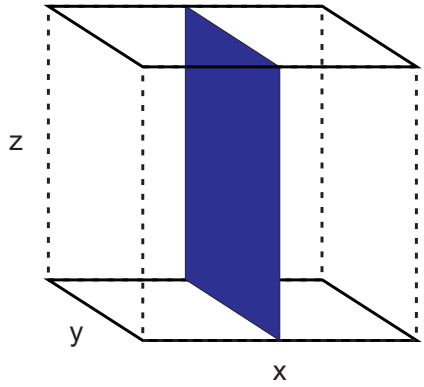
Jelikož je v jednotlivých implementacích zápis spojen s přesunem dat z pole vypočtených hodnot do pomocného pole pro zápis, je vedle času pro samotný zápis změřena také doba potřebná pro přesun dat do pole pro zápis. Protože přesun dat je nedílnou součástí všech typů zápisu, jsou při výpočtu zrychlení porovnány rychlosti vypočtené z časů zahrnujících přesun dat.

## 4.1 Samplování na desce kolmé k dekompoziční rovině

V prvním testovaném případě se jedná o domény, ve kterých je umístěna jedna nebo malé množství subdomén o rozměrech 1 x 512 x 512 (subdomény o těchto rozměrech budou nadále označovány jako desky). Tyto subdomény jsou umístěny tak, že přidělují všem procesům stejné, avšak malé množství dat, maximálně malé desítky kilobajtů. Příklad umístění jedné subdomény tímto způsobem je naznačen na Obrázku 4.1.

Pokud bychom pro simulaci využili 256 procesů, což je v současnosti typický počet procesů využívaný při simulacích v k-Wave, na každý proces připadnou dva řádky z každé desky, což odpovídá čtyřem kilobajtům. Takovéto malé množství dat na procesy způsobí při nativním způsobu zápisu rychlost 1 111,13 MB/s. Tuto rychlost se podařilo překonat všemi optimalizujícími metodami. Abychom srovnali jejich chování, byl proveden test slabého a silného škálování.

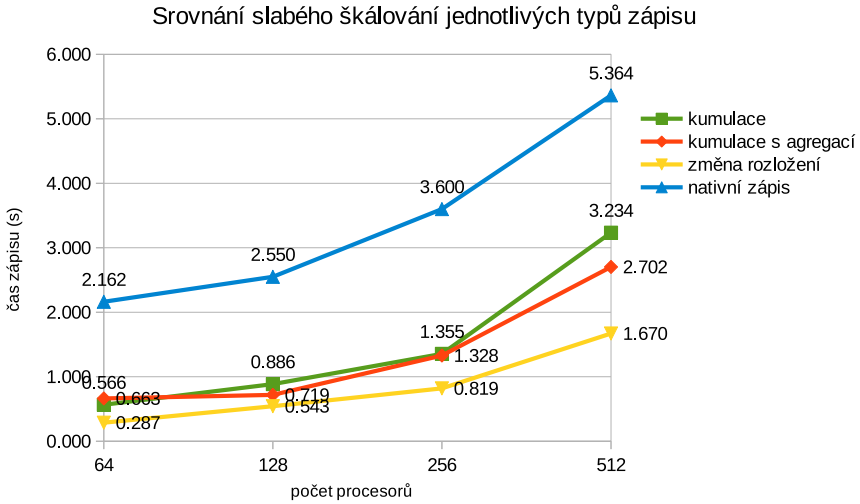
Nejprve se podívejme na slabé škálování metod. Spouštíme testované případy s různým počtem procesů, přičemž spolu s počtem procesů roste stejnou mírou také velikost problému,




Obrázek 4.1: Jediná subdoména uvnitř simulační domény přidělující všem procesům stejné, ale malé množství dat na zápis.

v tomto případě zapisovaných dat. Jestliže zvyšujeme oba parametry ekvivalentně, pak by v ideálním případě měla zůstat doba provádění stejná jako v předchozím případě.

Na Grafu 4.2 jsou vyobrazeny výsledky testů, kde testování začínalo použitím 64 procesů, které zápisovaly jedinou deskovou subdoménu. Množství zapisovaných dat dále rostlo spolu s počtem procesů, tudíž 128, 256 a 512 procesů zapisovalo data ze dvou, čtyř, respektive osmi subdomén. Při všech těchto testech připadlo na každý proces v jedné iteraci právě šestnáct kilobajtů zapisovaných dat.




Graf 4.2: Graf srovnávající jednotlivé typy zápisů skrze slabé škálování.

Z grafu lze vidět, že žádný z typů zápisu neškáluje ideálním způsobem. Jestliže srovnáme zápis jedné a čtyř subdomén, čas se zhoršil nejméně v případě nativního zápisu (1,67x), zatímco nejhůře v tomto ohledu dopadla změna rozložení zapisovaných dat (2,85x). Avšak časový rozdíl mezi těmito typy zápisu je propastný.

Přibližme si chování jednotlivých metod na příkladu testů s 256 procesy a třemi deskami. Pří optimalizaci prvním způsobem, tedy pouhou kumulací dat, došlo ke kumulaci přes

64 iterací s cílem zvětšit zapisovaná data v každém procesu na jeden megabajt. Druhá verze optimalizace zredukovala počet zapisujících procesů na jednu osminu, čímž se v jednotlivých procesech zvedlo množství dat na 128 kB a pro toto množství dat bylo I/O analýzou určeno, že ideální počet kumulujících iterací je deset. Celkově se tedy tímto způsobem zvedlo množství dat na zapisující procesy osmdesátkrát až na 1,25 MB. Metoda změny rozložení dat postupovala stejně jako předchozí, avšak nedošlo k redukci počtu zapisujících procesů, tudíž namísto 32 procesů zapisovalo opět 256.

Pokud budeme srovnávat výsledné rychlosti jednotlivých typů zápisů, tak zjitíme, že všechny metody dosáhly lepších výsledků než nativní zápis. V Tabulce 4.1 jsou časy přepočteny na rychlost zápisu a je také uvedeno, jaké minimální a maximální zrychlení oproti nativnímu zápisu bylo danou metodou dosaženo. Jednoznačně zjišťujeme, že optimalizace změnou rozložení dat pro zápis získáme nejlepší výsledky. Nejvyšší rychlosti bylo vždy dosaženo právě takto optimalizovaným zápisem, který při spuštění s 256 procesy atakoval hranici pěti gigabajt za sekundu, zatímco předchozí metody těsně nedosáhly na rychlost tří gigabajtů za sekundu.

| # procesů | nativní zápis | kumulace | kumulace s agregací | změna rozložení |
|---|---|---|---|---|
| **64** | 462,51 | 1 767,21 | 1 507,38 | 3 479,57 |
| **128** | 784,44 | 2 258,36 | 2 783,45 | 3 686,29 |
| **256** | 1 111,13 | 2 952,52 | 3 011,19 | 4 883,65 |
| **512** | 1 491,46 | 2 473,61 | 2 960,80 | 4 790,58 |
| zrychlení | | 1,66 – 3,82 | 1,99 – 3,55 | 3,21 – 7,52 |

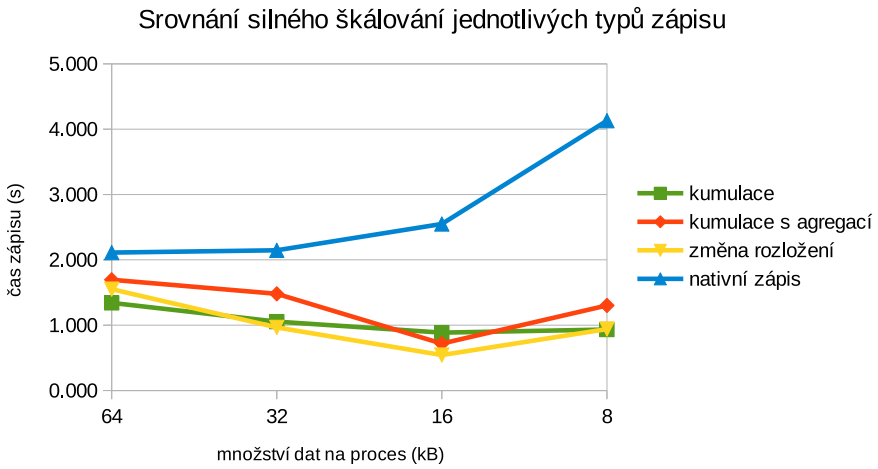
Tabulka 4.1: Rychlost zápisu (MB/s) naměřená během měření škálování spolu s uvedeným minimálním a maximálním zrychlením nativního zápisu. Zápis, který vedl k největšímu zrychlení je označen červeně, nejmenšímu modře.

Pro úplnost je zde uveden také test silného škálování, ve kterém jsme doménu obsahující dvě desky rozdělili mezi 32 procesů. Následně počet procesů na stejnou doménu stoupal. V zcela ideálním případě by mělo s dvojnásobným počtem procesů dojít ke zkrácení času na polovinu. Jak můžeme vidět v Grafu 4.3, dobu provádění mírně snížily pouze optimalizované metody zápisu s rostoucím počtem procesů, přičemž čas na nativní zápis rostl.

Žádný ze zápisů nepřekonal rychlost čtyř gigabajtů za sekundu, což je však dáno především tím, že s rostoucím počtem procesů klesá množství dat na každý z nich (z počátečních 64 kB až na 8 kB dat na zápis na proces), tudíž snižuje také rychlost zápisu. Zcela jasně je vidět pokles rychlosti se snižováním dat na rychlostech nativního zápisu v Tabulce 4.2.

| # procesů | nativní zápis | kumulace | kumulace s agregací | změna rozložení |
|---|---|---|---|---|
| **32** | 947,51 | 1 491,22 | 1 180,28 | 1 289,38 |
| **64** | 931,92 | 1 897,20 | 1 351,18 | 2 070,26 |
| **128** | 784,44 | 2 258,36 | 2 783,45 | 3 686,29 |
| **256** | 484,25 | 2 142,42 | 1 536,32 | 2 126,76 |
| zrychlení | | 1,57 – 4,42 | 1,25 – 3,55 | 1,36 – 4,70 |

Tabulka 4.2: Rychlost zápisu (MB/s) naměřená během měření silného škálování spolu s uvedeným minimálním a maximálním zrychlením nativního zápisu. Zápis, který vedl k největšímu zrychlení je označen červeně, nejmenšímu modře.




Graf 4.3: Graf silného škálování jednotlivých metod zápisu.

## 4.2 Různá rozložení stejného množství zapisovaných dat

Následující test umisťuje do domény tři subdomény tvaru desky, budou tedy rozebrány způsoby zápisu tří megabajtů dat na jednu iteraci. Pokud jsou tyto subdomény umístěny stejně jako v předchozím sekci 4.1, označme jej jako případ#1, tedy způsobem přidělujícím data na zápis všem procesům rovnoměrně, již víme, že optimalizovaný zápis pomocí transformace rozložení dat přinese zrychlení. Nyní nás však zajímá rychlost zápisu, jestliže umístíme tyto subdomény jiným způsobem. Cílem optimalizace je dosáhnout maximální možné rychlosti, ale také odstínit způsob rozložení dat tak, aby se to na rychlosti pokud možno neodrazilo.

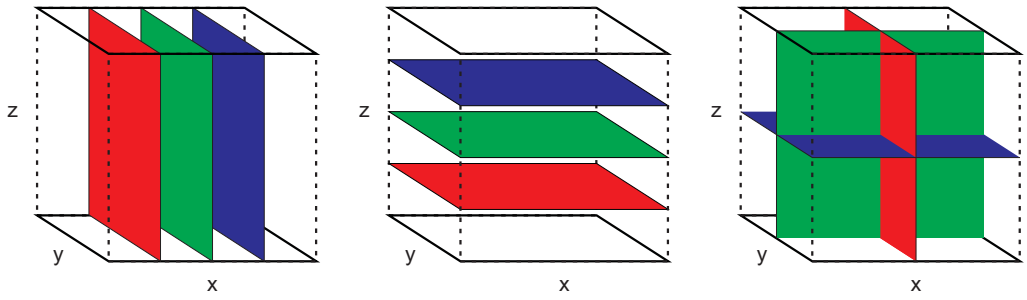
Nejprve se podíváme na tři subdomény kolmé k ose $z$, která je rozdělena mezi procesy (případ#2). Tímto způsobem přidělíme zapisujícím procesům jeden megabajt, což je velké množství dat ve srovnání s čtyřiceti osmi kilobajty, které připadly na každý z 256 zapisujících procesů v případě#1. Nyní bude využito 256 procesů znovu, zásadním faktorem však je, že data pro zápis mají nyní jen tři procesy.

V tomto případě bude optimalizace kumulací shodná s nativním zápisem, jelikož na procesy připadá dostatek dat. Metoda s agregací procesů v tomto případě nebude snižovat počet procesů, ale jak z I/O analýzy vyplývá, i pro toto množství dat je výhodné provádět malé množství kumulujících iterací. Proto v tomto případě došlo ke kumulaci dat přes devět iterací. Tímto způsobem se z rychlosti nativního zápisu 1 118,08 MB/s rychlost zvýšila na 3 359,72 MB, přičemž rychlost samotného zápisu byla změřena na 5 129,14 MB/s.

Komplementárním k předchozím umístěním subdomén je doména obsahující tři desky, které jsou ve středech jednotlivých dimenzí [1]. Tento případ je vyobrazen na Obrázku 4.4 jako případ#3. Jedná se tak o třetí možný způsob rozložení stejného množství dat. V tomto případě jsou data rozložena mezi všechny procesy rovnoměrně, vyjma jediného, jež spravuje desku kolmou na rovinu dekompozice.

Základním předpokladem je vyrovnat množství dat na zapisující procesy. Postavil jsme proto vedle nativního zápisu opět nativní zápis, kterému však předchází proces vyrovnání. Samotné vyrovnání dat na zapisující procesy mělo překvapivě malý vliv na rychlost zápisu,

[1] Uvedené řezy středy dimenzí jsou tři základní řezy v medicíně: transversální, sagitální, koronální.




Obrázek 4.4: Vyobrazení různých umístění tří subdomén rozdělujících data na zápis rovnoměrně mezi všechny procesy (případ#1), rovnoměrně, ale pouze pro tři procesy (případ#2) nebo mezi všechny procesy, ale nerovnoměrně (případ#3).

jak je možné vyčíst z Tabulky 4.3, ale přesto je nedílnou součástí metody optimalizace změnou rozložení dat, aby bylo možné určit jednotný počet kumulujících iterací pro všechny skupiny procesů, které tato metoda vytváří.

Metoda změny rozložení vytvořila tři skupiny procesů, ve kterých byl vždy určen jediný zapisující proces (konkrétní postup tvorby skupin je popsán v sekci 3.3.3 věnující se této metodě). Jedna skupina je tvořena procesem, na který připadla subdoména kolmá na osu $z$, ve zbylých skupinách vybrané procesy přebírají data všech okolních procesů. Tímto postupem byl případ#3 převeden na případ#2, který po optimalizaci vykazuje rychlost zápisu 3 359,72 MB/s. Bohužel této rychlosti se nepodařilo dosáhnout, jelikož se zastavila na úrovni 2 816,2 MB/s. Tato rychlost je výrazně ovlivněna především přesunem velkého množství dat k zapisujícím procesům, a to z rozsáhlé skupiny procesů.

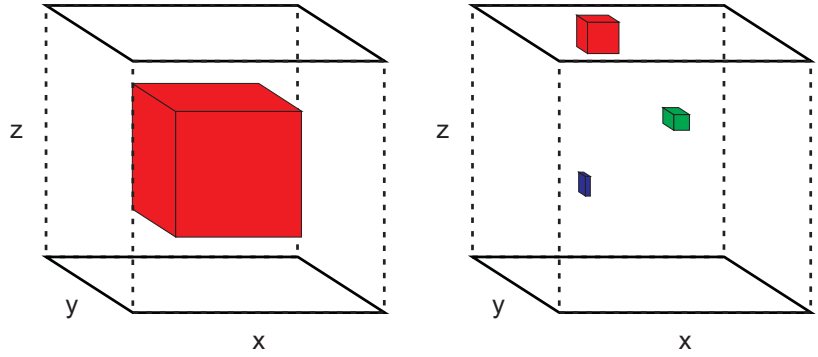
| | nativní zápis | změna rozložení | vyrovnání dat |
|---|---|---|---|
| **případ#1** | 1 227,12 | 5 254,7 | |
| **případ#2** | 1 118,08 | 3 359,72 | |
| **případ#3** | 708,976 | 2 816,2 | 757,026 |

Tabulka 4.3: Rychlost zápisu (MB/s) různého rozložení tří subdomén při vyžití 256 procesů.

## 4.3 Kvádrové domény

V následujících testech, které jsou vykresleny na Obrázku 4.5, nejsou uvnitř domény subdomény ve tvaru desek, ale jedná se o různé velikosti kvádrů.

Prvním z těchto testů je případ#4, ve kterém je uprostřed domény umístěná kostka o rozměrech 256 x 256 x 256, která přiděluje data na zápis polovině z celkového počtu procesů. Jedná se o optimální případ pro nativní způsob zápisu, jelikož data zapisuje mnoho procesů s poměrně velkým množstvím dat. Jestliže provedeme test s 256 procesy, připadne na každý zapisující proces polovina megabajtu. Těmto příznivým podmínkám odpovídá také rychlost nativního zápisu, jež přesáhla rychlost čtyř gigabajtů za sekundu (více v Tabulce 4.4), což je hodnota více než dvakrát vyšší než ve všech předchozích testovaných případech.




Obrázek 4.5: Domény se subdoménami ve tvarech kvádrů, případ#4 a případ#5.

Jelikož v případě#4 je přiděleno dostatečné množství dat, metoda kumulace prováděla zápisy vždy po dvou iteracích. Tato změna nativního zápisu však zrychlení nepřinesla. Také metodou kumulace s agregací dat okolních procesů se tato rychlost bohužel nepodařila zvýšit, avšak zůstala na přibližně stejné úrovni jako rychlost nativního zápisu. Zásadní zrychlení však přináší optimalizace změnou rozložení dat, která dosáhla až na 13 976,85 MB/s. Tato rychlost odpovídá rychlosti dosažené při testování maximální možné rychlosti zápisu při využití sto dvaceti osmi procesů zapisujících osm megabajtů na proces (viz sekce 3.2.3, Graf 3.6). Zároveň v porovnání s maximální rychlostí zápisu, které bylo dosaženo při zápisu třiceti dvou megabajtů na proces, je tato hodnota o pouhých 16 % nižší.

Opačným případem vůči předchozímu je druhé rozložení kvádrových subdomén v doméně, označené jako případ#5. Tento test je tvořen subdoménami o rozměrech 64 x 64 x 64, 33 x 65 x 33 a 10 x 40 x 41, které jsou rozmístěny různě po doméně, takže žádný z procesů nezapisuje data z více subdomén. Vznikly tak tři skupiny procesů s malým a navíc různým množstvím zapisovaných dat (32, 16 a 3 kB při využití 256 procesů). Tato skutečnost se odrazila na rychlosti nativního zápisu, která byla 457,44 MB/s.

Metoda kumulace dat určila pro toto rozložení dat jako nejvhodnější šedesát tři kumulujících iterací, tedy vysoký počet. To se však nikterak neodrazilo na rychlosti zápisu, který zůstal přibližně na stejné úrovni. Oproti tomu po vyrovnání rozdílů v množství dat a kumulaci dat přes 15 iterací došlo k zvýšení rychlosti na 3 036 MB/s, což je v porovnání s rychlostí nativního zápisu zrychlení 6,6násobné. Toto zrychlení výrazně předčilo také zrychlení dosažené metodou změny rozložení, která však bylo omezena počtem zapisujících procesů ve skupině procesů s daty nejmenší domény. Počet zapujích procesů této skupiny byl prvně zredukován postupem vyrovnání množství dat na zapisující procesy, tudíž v této skupině tak zůstaly pouze dva zapisující procesy, což evidentně nepřineslo požadovaný efekt.

| | nativní zápis | kumulace | kumulace s agregací | změna rozložení |
|---|---|---|---|---|
| **případ#4** | 4 348,96 | 1 320.27 | 4 295,46 | 13 976,85 |
| **případ#5** | 457,44 | 421.79 | 3 036,29 | 1 391,77 |

Tabulka 4.4: Rychlost zápisu (MB/s) testu se subdoménami ve tvaru kvádru prováděných 256 procesů.

## 4.4 Kuboidy

Vedle metod optimalizujících rychlost zápisu vznikla také implementace nahrazující lineární sensorovou masku, která označuje indexy v doméně, které chceme uložit, za kuboidy obalující tyto oblasti. Z hlediska rychlosti zápisu je kritickým případem, když doména obsahuje více subdomén, jelikož se jejich zápis musí serializovat.

Abych změřil vliv serializace zápisu několika subdomén, zopakoval jsem test s případem#1 a případem#2, kdy jsou v doméně tři desky, a využil jsem k tomu 256 procesů. Z důvodu serializace se dá očekávat, že rychlost klesne ekvivalentně s počtem subdomén, a zároveň musíme započítat stejný pokles množství zapisovaných dat na proces. Pro tyto případy tak lze rychlost zápisu odhadnout jako třetinovou oproti zápisu jedné desky pomocí 256 procesů. Srovnání rychlosti tohoto zápisu se zápisem pomocí kuboidů je zaznamenáno v Tabulce 4.5.

Pro případ#1, kdy subdomény přidělují data všem procesům, byla změřena rychlost zápisu do kuboidu 112,73 MB/s, což přesně koresponduje s rychlostí nativního zápisu stejně umístěné jediné desky, která dosáhla 114,88 MB/s. Tento zápis se podařilo dvacetkrát zrychlit na rychlost 2 336,97 MB/s pomocí pouhé kumulace dat, která však především zajistila, že se každý kuboid zapisoval v jiné iteraci, tudíž nedocházelo ke vzájemnému blokování.

Jestliže došlo k otočení subdomén do pozic případu#2, klesla rychlost na pouhých 4,7 MB/s, avšak i třetina rychlosti nativního zápisu do takto nastavené subdomény je jen 26,16 MB/s. Tyto hodnoty jsou vůbec nejmenšími změřenými rychlostmi ze všech testů. Potěšující je dvacetinásobné zrychlení, kterého se podařilo optimalizací dosáhnout, nicméně to stále znamená rychlost jen 83,11 MB/s.

Vedle případů, které obsahovaly vícero subdomén, a tak očekávaně způsobovaly nízkou rychlost zápisu při využití kuboidů, jsem otestoval také situaci s jedinou subdoménou s mnoha zapisujícími procesy i daty na zápis, tedy případ#4. Opět jsem využil 256 procesů. Tato situace je ideální pro nativní zápis, avšak při využití kuboidů tato rychlost výrazně klesla až na 430,13 MB/s. Pomocí kumulace přes deset iterací se však i tuto nízkou rychlost podařilo pětkrát zvýšit na 2 115,47 MB/s.

| | nativní zápis | kuboid | kuboid s kumulací | $\frac{1}{3}$ nativního zápisu 1 desky |
|---|---|---|---|---|
| **případ#1** | 1 118,08 | 112,73 | 2 336,97 | 114,88 |
| **případ#2** | 1 227,12 | 4,7 | 83,11 | 26,16 |
| **případ#4** | 4 348,96 | 430,13 | 2 115,47 | |

Tabulka 4.5: Rychlost zápisu (MB/s) pomocí kuboidů, kuboidů optimalizovaných kumulací, spolu s odpovídajícími neoptimalizovanými zápisy s využitím lineární sensorové masky.

# Kapitola 5

# Závěr

Cílem této práce bylo vyvinout a implementovat nový, rychlejší způsob zápisu pro aplikaci k-Wave. Tento I/O subsytém řeší především problém mnoha zápisů malého množství dat, proto je založen na zvětšení množství zapisovaných dat na jednotlivé procesy pomocí kumulace přes několik iterací a na přeskupení dat mezi procesy.

Pro dosažení nejlepšího řešení byl zmapován vstupně-výstupní systém superpočítačů a řada doporučení a způsobů jak zápis optimalizovat. Jelikož k-Wave vyžaduje nastavení veškerých pomocných bufferů na počátku běhu aplikace, byla vyvinuta aplikace provádějící I/O analýzu cílového počítače. Aplikace k-Wave tuto analýzu načítá a nastavuje dle ní nejvýhodnější hodnoty parametrů.

Na základě zjištěných možností byly navrženy tři algoritmy optimalizace zápisu, kombinující různými způsoby kumulaci a agregaci dat. Všechny metody byly implementovány do aplikace k-Wave, otestovány na řadě testů a porovnány s nativním zápisem i mezi sebou. Všechny navržené metody vedly k rychlejšímu zápisu, přičemž nejlepší naměřené výsledky ve většině případů produkoval zápis pomocí metody změny rozložení zapisovaných dat. Tato metoda ve všech testovaných případech přinesla zrychlení a například při testu slabého škálování vždy zrychlila nativní zápis minimálně 3,2krát a nejvíce až 7,5krát. Vedle sady testů, které se vyznačují pomalým nativním zápisem, byl postaven také test s ideálními podmínkami pro nativní zápis. I při tomto testu došlo ke zrychlení zápisu 3,2krát, čímž byla dosažena rychlost 13,65 GB/s, která atakuje teoretickou maximální rychlost.

Vedle optimalizací zápisu byla vytvořena také verze k-Wave podporující označování subdomén pomocí rohů kvádrů obalujících oblast zájmu. Zavedení takzvaných kuboidů přináší výhody pro následné zpracování dat, kdy je umožněno rychlejší načítání souboru po blocích, avšak má velmi negativní vliv na rychlost zápisu. Pomocí optimalizace kumulací, která desynchronizuje blokující zápisy do jednotlivých subdomén, je zápis výrazně optimalizován. Nicméně ani po optimalizaci zdaleka nedosahuje rychlosti optimalizovaného zápisu využívajícího sensorovou masku.

Tabulky s grafy srovnávající všechny testované případy jsou umístěny v příloze B. V přloze je umístěn také poster A.1, se kterým jsem prezentoval výsledky této práce na konferenci Excel@Fit 2016.

Nová implementace přináší významnou úsporu výpočetního času a s tím spojenou finanční náročnost simulace. Veškeré metody jsou navíc použitelné ve všech superpočítačových aplikacích s distribouvanými daty na zápis, který probíhá v jednotlivých iteracích běhu dané aplikace.

Tuto práci je možné v budoucnu rošířit. Poté, co vznikne nová verze knihovny HDF5 s implementací neblokujícího kolektivního zápisu, bude velmi zajímavé vyzkoušet modifiko-

vat novou optimalizovanou verzi zápisu pomocí těchto neblokujících funkcí. V současnosti se také vyvíjí systém komprese dat, který působí opačným efektem vůči zvětšování dat na zapisující procesy, což znamená, že bude nutné tyto změny v I/O susbsystému reflektovat.

Na vytvořené soubory se také vztahuje potřeba šifrování dat, aby došlo k zajištění bezpečnosti především citlivých osobních medicínských informací, která by měla být chráněny před zneužitím. Bylo by velmi časově náročné šifrovat velké, mnoha gigabajtové (případně terabajtové) soubory, které jsou výstupem simulace. Zde se projevuje další z výhod využití HDF5, protože namísto šifrování celého datového souboru stačí provést zašifrování metadat, což je soubor o velikosti několika kilobajtů, bez kterého není možné daný soubor číst, nebo s ním jakkoli jinak pracovat.

# Literatura


[1] Allinea Software Ltd.: Allinea DDT: The debugger for C, C++ and F90 threaded and parallel code. [online], [cit. 2016-05-15].
URL http://www.allinea.com/products/ddt

[2] Allinea Software Ltd.: Allinea MAP - C/C++ profiler and Fortran profiler for high performance Linux code. [online], [cit. 2016-05-15].
URL http://www.allinea.com/products/map

[3] Altair: PBS Proffesional. [online], 2016, [cit. 2016-05-23].
URL http://www.pbsworks.com/Product.aspx?id=1

[4] Argonne National Laboratory: ROMIO. [online], [cit. 2015-12-27].
URL http://press3.mcs.anl.gov/romio/

[5] Cluster File Systems, Inc.: Lustre: A Scalable, High-Performance File System. 2002.
URL http://www.cse.buffalo.edu/faculty/tkosar/cse710/papers/lustre-whitepaper.pdf

[6] Goldstein, S. C.: Parallel Architecture Fundamentals. [online], 2003, [cit. 2015-10-20].
URL http://www.cs.cmu.edu/afs/cs/academic/class/15740-f03/www/lectures/lect08.4up.pdf

[7] Irish Centre for High-End Computing: Using DDT :: Distributed debugging tool. [online], [cit. 2016-05-15].
URL https://www.ichec.ie/support/tutorials/ddt

[8] IT4Innovations: IT4Innovations, národní superpočítačové centrum. [online], [cit. 2016-04-30].
URL https://www.it4i.cz/

[9] Jaros, J., Treeby, E. B., Cox, B. T., Nandapalan, N., Johnston, B. and Rendell, A. P.: Large-scale Ultrasound Simulations in Human Body. 2014, prezentace předmětu Architektura a programování paralelních systémů, Fakulta informačních technologií VUT Brno.

[10] Kaminsky, A.: *BIG CPU, BIG DATA: Solving the World's Toughest Computational Problems with Parallel Computing*. 2015, 123–144 s.
URL https://www.cs.rit.edu/~ark/bcbd/bcbd.pdf

[11] Kato, J.; Ishikawa, Y.: Design and Implementation of Parallel File Aggregation Mechanism. In *Proceedings of the 1st International Workshop on Runtime and Operating Systems for Supercomputers*, ROSS '11, New York, USA: ACM, 2011,

ISBN 978-1-4503-0761-1, s. 33–40, doi:10.1145/1988796.1988802.
URL http://doi.acm.org/10.1145/1988796.1988802

[12] Kogge, P.; Shalf, J.: Exascale Computing Trends: Adjusting to the "New Normal" for Computer Architecture. In *Computing in Science and Engineering*, Nov 2013, s. 16–26, doi:10.1109/MCSE.2013.95.

[13] Koziol, Q.: Parallel HDF5. [online], 2015-04-03.
URL http://slideplayer.com/slide/6604205/

[14] Lampa, P.: Procesy a vlákna. 2014, prezentace předmětu Pokročilé operační systémy, Fakulta informačních technologií VUT Brno.

[15] Latham, R., Ross, R., Welch, B. and Antypas, K.: Parallel I/O in Practice. [online].
URL https://www.nersc.gov/assets/Training/pio-in-practice-sc12.pdf

[16] Meglicki, Z.: High Performance Data Management and Processing, I590, Section 7462, MPI IO. [online], 2004-04-29, [cit. 2016-05-18].
URL http://beige.ucs.indiana.edu/I590/node86.html

[17] Message Passing Interface Forum: MPI: A Message-Passing Interface Standard, Version 3.1. 2015-06-04.
URL https://www.mpi-forum.org/docs/mpi-3.1/mpi31-report.pdf

[18] NASA: Hierarchical Data Format. [online], 2012-04-12, [cit. 2015-12-29].
URL https://eosweb.larc.nasa.gov/HBDOCS/hdf.html

[19] Nitzberg, B.; Lo, V.: Collective buffering: Improving parallel I/O performance. In *High Performance Distributed Computing, 1997. Proceedings. The Sixth IEEE International Symposium on*, Aug 1997, ISSN 1082-8907, s. 148–157, doi:10.1109/HPDC.1997.622371.

[20] Ohta, K., Kimpe, D., Cope, J., Iskra, K., Ross, R., Ishikawa, Y.: Optimization Techniques at the I/O Forwarding Layer. 2010.
URL http://www.mcs.anl.gov/papers/iofsl-cluster10.pdf

[21] OpenSFS, EOFS: Lustre. [online], 2015, [cit. 2015-12-27].
URL http://lustre.org/

[22] Oracle and Intel Corporation: Lustre Software Release 2.x: Operation manual: s. 6–7. [cit. 2015-12-27].
URL https://build.hpdd.intel.com/job/lustre-manual/lastSuccessfulBuild/artifact/lustre_manual.pdf

[23] Panasas, Inc: pNFS. [online], [cit. 2015-12-31].
URL http://www.pnfs.com/

[24] PRACE: Partnership for advanced computing in Europe. [online], [cit. 2016-04-30].
URL http://www.prace-ri.eu/

[25] Rutman, N.: Rock-Hard Lustre: Trends in Scalability and Quality. [online], 2010, [cit. 2016-03-15].
URL http://lustre.org/

[26] Skinner, D. and Antypas, K.: Parallel Application Scaling, Performance, and Efficiency. 2010.
URL https://www.nersc.gov/assets/NUG-Meetings/IntroMPIApplicationScaling.pdf

[27] Slíva, R.: HPC Users' Access Workshop: nový superpočítač Salomon. [online], [cit. 2015-12-31].
URL http://prace.it4i.cz/sites/prace.it4i.cz/files/files/salomon-09-2015-sliva.pdf

[28] Smrčka, A.: Úvod a pojmy v testování. 2014, prezentace předmětu Testování a dynamická analýza, Fakulta informačních technologií VUT Brno.

[29] Steve Fingerhut: Does Storage break Moore's Law? [online].
URL http://itblog.sandisk.com/does-storage-break-moores-law/

[30] Thakur, R.; Gropp, W.; Lusk, E.: Data sieving and collective I/O in ROMIO. In *Frontiers of Massively Parallel Computation, 1999. Frontiers '99. The Seventh Symposium on the*, Feb 1999, s. 182–189, doi:10.1109/FMPC.1999.750599.

[31] The HDF Group: HDF5 User's Guide. [online], 2015-11, [cit. 2015-01-10].
URL https://www.hdfgroup.org/HDF5/doc/UG/HDF5_Users_Guide-ResponsiveHTML5/index.html

[32] The HDF Group: The HDF5 Home Page. [online], 2015-11-12, [cit. 2015-12-29].
URL http://www.hdfgroup.org/HDF5

[33] The HDF Group: HDF5 Tools. [online], 2016-05-05, [cit. 2016-05-20].
URL http://www.hdfgroup.org/HDF5/doc/RM/Tools.html

[34] Thomason, L.: TinyXml Main Page. [online], [cit. 2016-04-30].
URL http://www.grinninglizard.com/tinyxml/

[35] Top500.org: Top500 Supercomputer Sites. [online], [cit. 2015-10-20].
URL http://www.top500.org/list/2015/06/

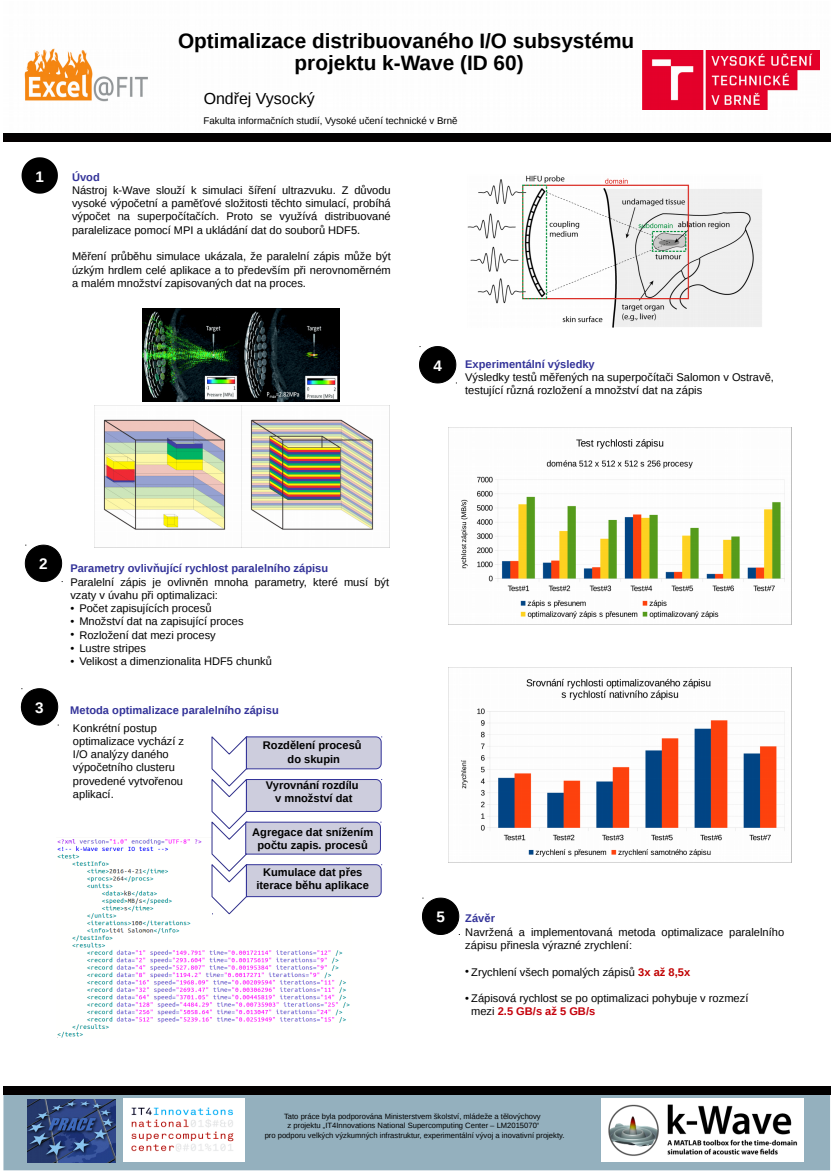
[36] Treeby, B. and Cox, B.: k-Wave A MATLAB toolbox for the time-domain simulation of acoustic wave fields. [online], [cit. 2014-03-19].
URL http://www.k-wave.org

[37] Unidata: netCDF. [online], [cit. 2015-12-31].
URL http://www.unidata.ucar.edu/software/netcdf/

[38] Vysocký, O.: Optimalizace I/O subsystému projektu k-Wave. 2014.
URL http://www.fit.vutbr.cz/study/DP/BP.php.cs?id=16703&file=t

# Příloha A

# Poster




Obrázek A.1: Poster vytvořený pro konferenci Excel@Fit 2016.

# Příloha B

# Grafy srovnání metod zápisu

Označení metod zápisu:

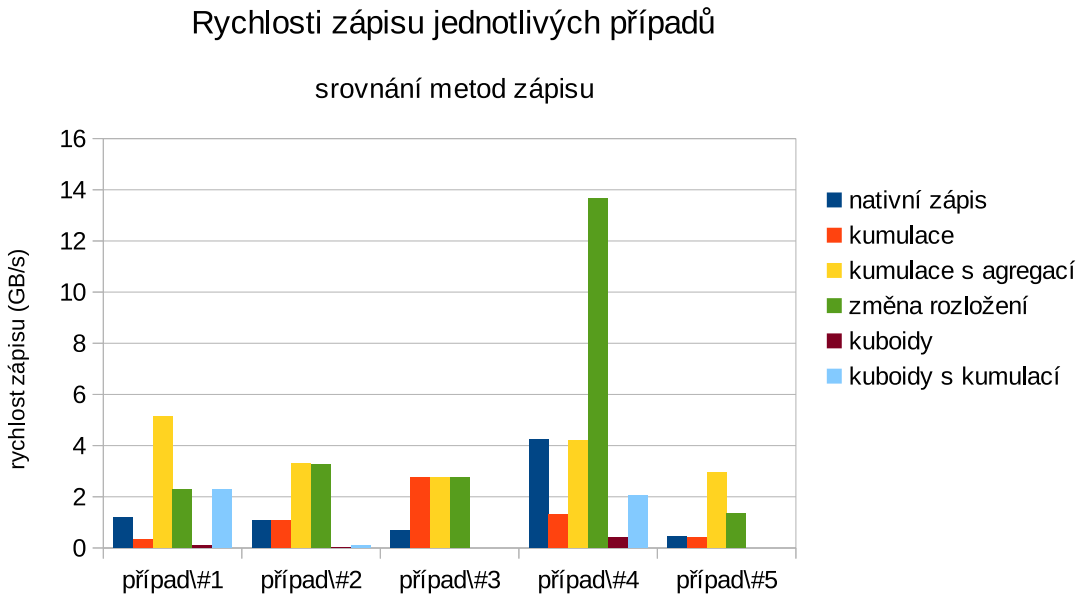
- M0 - nativní zápis
- M1 - kumulace
- M2 - kumulace s agregací
- M3 - změna rozložení
- M4 - kuboidy
- M5 - kuboidy s kumulací

| | Celková data | M0 | M1 | M2 | M3 | M4 | M5 |
|---|---|---|---|---|---|---|---|
| případ#1 | 3000 | 1227,12 | 340,54 | 5254,7 | 2329,87 | 112,73 | 2336,97 |
| případ#2 | 3000 | 1118,08 | M0 | 3359,72 | M2 | 4,27 | 83,11 |
| případ#3 | 3000 | 708,98 | M2 | 2816,2 | M2 | | |
| případ#4 | 64000 | 4348,96 | 1320,27 | 4295,46 | 13976,85 | 430,13 | 2115,47 |
| případ#5 | 1333 | 457,44 | 421,79 | 3036,29 | 1391,77 | | |

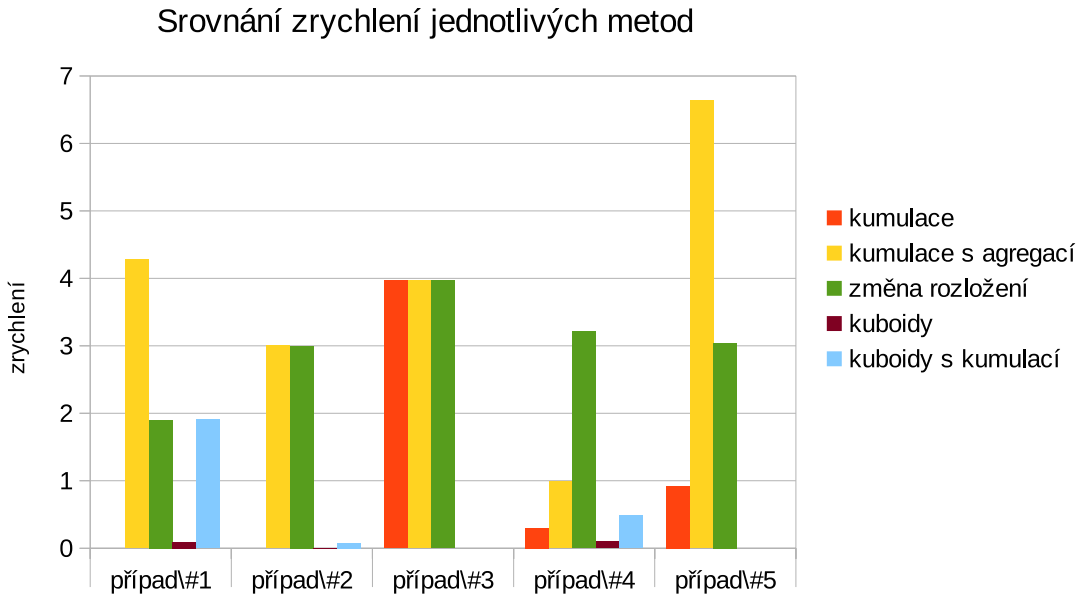
Tabulka B.1: Rychlost zápisu (MB/s) jednotlivých testovaných případů při vyžití 256 procesů. Zkratka názvu některé z metod v poli značí, že chování dané metody v tomto případě kopírovalo chování metody v poli.

| | M1 | M2 | M3 | M4 | M5 |
|---|---|---|---|---|---|
| případ#1 | 0,28 | 4,28 | 1,90 | 0,09 | 1,90 |
| případ#2 | 0,00 | 3,00 | 3,00 | 0,00 | 0,07 |
| případ#3 | 3,97 | 3,97 | 3,97 | | |
| případ#4 | 0,30 | 0,99 | 3,21 | 0,10 | 0,49 |
| případ#5 | 0,92 | 6,64 | 3,04 | | |

Tabulka B.2: Zrychlení jednotlivých testovaných metod vůči nativnímu zápisu při vyžití 256 procesů.




Graf B.1: Srovnání rychlosti zápisu jednotlivých metod v testovaných případech s využitím 256 procesů.




Graf B.2: Srovnání zrychlení zápisu jednotlivých metod v testovaných případech vůči nativnímu zápisu s využitím 256 procesů.