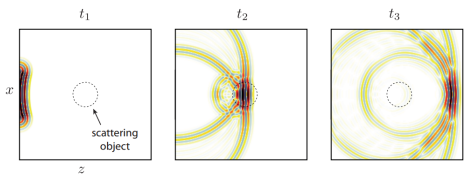
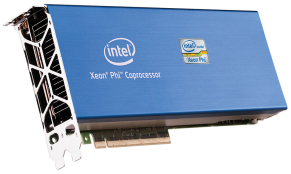
Excel@FIT 2015
http://excel.fit.vutbr.cz

Implementácia výpočtovo náročných algoritmov na Intel Xeon Phi

Dominik Šimek*





Abstrakt
Motiváciou pre vznik tejto práce bolo nasadenie a optimalizácia výpočtovo náročných algoritmov na koprocesor Intel Xeon Phi. Tabuľkovo koprocesor Intel Xeon Phi disponuje podstatne väčším výkonom ako klasický procesor, preto sa javí ako veľmi zaujímavá architektúra. Celkovo je táto technológia veľmi mladá a ešte neprebádaná, preto stojí za úsilie priniesť pohľad, ktorý by zjednodušil prácu s koprocesorom a umožnil jeho efektívne využívanie. Cieľom tohto dokumentu je stručne oboznámiť čitateľa s mojou prácou, ktorá by mohla byť v blýzkej budúcnosti veľmi zaujímavá a využiteľná.
Na koľko je Intel Xeon Phi mladou technológiou, jeho efektívne využívanie je pomerne náročné. Práca sa teda javí ako veľká výzva, ktorú by bola škoda nevyužiť. Práca s koprocesorom Intel Xeon Phi má budúcnosť v obore High Performance Computing, kde by ho bolo možné používať napríklad na miesto akcelerátorov GPGPU. Vysoko optimalizované výpočtovo náročné algoritmy by teda mali bežať na tomto koprocesore veľmi rýchlo. Aká je ale pravda? Je vysoký výkon koprocesoru naozaj využiteľný?
V bakalárskej práci sme postupovali od jednoduchších benchmarkov k zložitejším problémom, ktoré sme sa snažili dôkladne vyhodnotiť. K zaujímavejším problémom riešených v tejto práci patrí napríklad simulácia pohybu častíc v priestore (N-Body), simulácia šírenia akustických vĺn v 1D, 2D a 3D, optimalizácia extrakcie ivectoru, ktorá sa využíva pri spracovaní reči.

Kľúčové slová: Intel Xeon Phi — Koprocesor — N-Body — k-Wave — HPC

Priložené materiály: Github repozitár s ukážkami zdrojových kód a výsledkami meraní

*xsimek23@stud.fit.vutbr.cz, *Faculty of Information Technology, Brno University of Technology*

1. Úvod

[Motivácia] Motivácia práce bola jasná – výstavba nového superpočítača v Ostrave (Salomon), ktorý bude okrem iného disponovať obrovským množstvom koprocesorov Intel Xeon Phi. Aktuálne veľmi využívaný superpočítač Anselm ponúka 4 akcelerátory Intel Xeon Phi. Salomon ich bude obsahovať až 864. Toto číslo hovorí samo za seba, preto nieje potrebné zdôrazňovať podstatu a využitie tejto práce.

[Definícia problému] Intel Xeon Phi sám o sebe ponúka viac ako 50 jadier, ktoré spolu dokážu vyvinúť slušný výkon. Na prvý pohľad to teda vyzerá jednoducho, máme jeden čip (s vysokým výkonom), ktorý obsahuje veľké množstvo jadier – super. Realita však môže byť mierne odlišná, keďže samotné 1 jadro produkuje žalostne malý výkon. Preto je veľmi dôležité využívať čo najviac jadier a algoritmus škáľovať na čo najviac vlákien.

Na začiatok bolo nutné dôkladne naštudovať architektúru, s ktorou zatiaľ pracovalo len veľmi málo ľudí. Samozrejme, teória nieje všetko, preto bolo nutné získať skúsenosti na základe jednoduchých benchmarkov, od ktorých sa neskôr prešlo k podstatne zložitejším problémom.

Nieje problém vytvoriť jednoduchý benchmark, ktorý pobeží (zatiaľ pravdepodobne neoptimálne) na karte Xeon Phi, postačí jednoduchý kód v jazykoch C, C++ alebo Fortran a Intel kompilátor. Je možné pracovať najmä s štandartnými knižnicami vyššie uvedených programovocích jazykov a knižnicou Intel MKL. Problémy môžu nastať pri tvorbe zložitejšej aplikácie, kde potrebujeme použiť napríklad neštandartnú knižnicu, ktorú treba prispôsobiť potrebám koprocesoru.

[Existujúce riešenia] Intel ponúka rôzne manuály, návody a príručky ohľadom práce s naším koprocesorom, ktoré pomôžu pri ťažkých začiatkoch. Veľmi dôležitou súčasťov je knižnica Intel MKL, ktorá ponúka veľké množstvo (nielen) matematických funkcií, ktoré sú vysoko optimalizované ako pre rôzne procesory, tak aj pre náš koprocesor. Koprocesor je na trhu od roku 2012, stále je ale používaný iba malým množstvom užívateľov. Pri náročných výpočtoch sa rozrastá najmä používanie akcelerátorov GPGPU, ktoré sú na trhu už dlhšie ako koprocesory Intel Xeon Phi, ich programovací model je teda podstatne viac zaužívaný. Po dôkladnom prehľadávaní webu je možné nájsť isté praktické príklady využitia Xeon Phi, nieje ich však veľké množstvo. Z tohto dôvodu je ťažké hovoriť o existújúcich riešeniach, preto je v tejto chvíli dôležité tvoriť, zdieľať vlastné nápady a skúsenosti, ktoré môžu pomôcť ďalším úžívateľom a tým zvýšiť využívanie tejto akceleračnej karty.

[Vlastné riešenia] Počas tvorby bakalárskej práce vzniklo docela veľké množstvo rôznych príkladov, benchmarkov, či návodov, ktoré môžu pomôcť pri práci s koprocesorom. Na žačiatku práce sa riešili jednoduché veci typu súčin matíc, matice a vektora a pod. Postupne sa prechádzalo na zložitejšie problémy, ako napríklad „N-Body Simulation“, teda simulácia pohybu častích v priestore, z ktorej sa podarilo dostať celkom slušný výkon. Veľmi zaujímavou príležitosťou bola práca na projekte k-Wave, ktorý rieši simuláciu šírenia ultrazvukových vĺn v mäkkých tkanivách. Jednalo sa o tisícky riadkov C++ kódu optimalizovaného pre procesory, ktorý bolo treba dostať na Xeon Phi, zmerať výkon a optimalizovať. Program používal neštandartné knihovne ako napríklad HDF5, ZLIB a pod., ktoré bolo nutné prispôsobiť potrebám koprocesoru.

Ďalšou veľkou výzvou bola kroskompilácia Pythonu a modulov Numpy a Scipy pre Xeon Phi. Python bežiaci na karte Xeon Phi bol určený pre výskumnú skupinu na FIT VUT, ktorá sa zaoberá spracovaním ľudskej reči.

Všetky tieto experimenty zabrali veľké množstvo času a vyžadovali by si ešte oveľa viac, preto nie vždy všetko dopadlo tak ako by sme chceli. O úspechoch a neúspechoch prechádzajúcich experimentov si povieme v ďalších kapitolách.

[Prínos práce] Počas práce s Xeon Phi som sa dozvedel veľké množstvo zaujímavých vecí, ktoré by som rád posunul ďalším záujemcom o prácu práve s týmto koprocesorom. Preto by mohlo byť zaujímavé, keby bakalárska práca po dokončení mohla slúžiť ako akási príručka, či už pri začiatkoch s Xeon Phi, alebo pri jeho pokročilom programovaní či spravovaní. Koprocesor Xeon Phi bol podrobený veľkému množstvu experimentov, za účelom identifikovať jeho silné či slabé miesta, vyhodnotiť jeho potenciál a možnosti využitia.

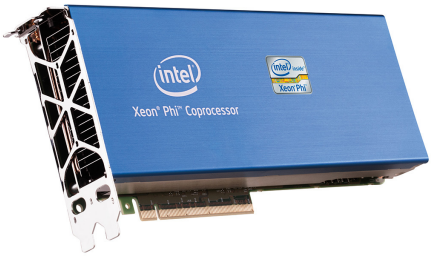
2. Teoretické základy

Na začiatok by bolo vhodné oboznámiť čitateľa, zo základnými princípmi a architektúrov koprocesoru Intel Xeon Phi. Pre optimalizáciu kódu a následné efektívne využitie koprocesoru je dôležité pochopiť ako systém funguje minimálne z hľadiska organizácie pamätí, vlákien a podobne.

2.1 História koprocesoru Intel Xeon Phi

Vývoj Intel Xeon Phi začal v období, kedy sa hľadalo riešenie pre zníženie spotreby energie procesorov Intel Xeon vyvinutých približne v roku 2001. Bola navrhnutá jednoduchá, nízkofrekvenčná MIC architektúra, ktorá bola schopná produkovať väčší výkon pri rozumnej spotrebe energie. Skratka MIC označuje technológiu vyvinutú firmou Intel – „Intel Many Integrated Core Architecture“, ktorá kombinuje veľké množstvo jadier procesorov Intel na jednom čipe.

Technológia MIC mala byť využiteľná v rôznych oblastiach, ako napríklad počítačová grafika, vedecké, technické aplikácie a všeobecne výpočtovo náročné algortimy. Inžiniery firmy Intel teda vyvinuli novú mikroarchitektúru založenú na jadrách Intelx86 (Pentium), ku ktorej bol vyvinutý operačný systém založený na jadre Linuxu. Prvý produkt dostal označenie KNC - „Knights Corner“, ktorý dosahoval výkon 2 teraFLOPS pre čísla s jednoduchou presnosťou, resp. 1 teraFLOPS pre čísla typu `double`. Koprocesor sa dostal na trh v roku 2012 po označením Intel Xeon Phi.



Obrázok 1. Intel Xeon Phi.

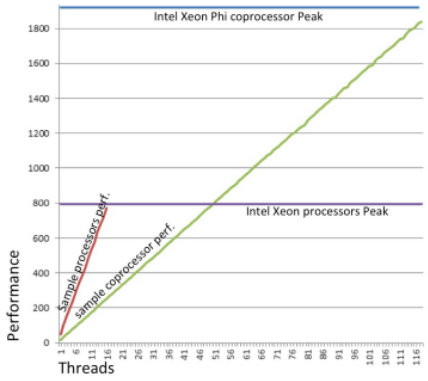
2.2 Základné parametre koprocesoru

Architekúta Intel MIC bola navrhnutá tak, aby poskytovala vysoký výkon a efektivitu pri zachovaní programovacieho prístupu procesorov Intel Xeon. Koprocesor je riadený operačným systémom Linux, podporuje usporiadanie pamäte x86, aritmetiku desatinnej čiarky IEEE 754 a je schopný behu aplikácií vytvorených v programovacích jazykoch C, C++ a Fortran. Samozrejme existuje podpora pre paralélne programovanie, či už na úrovni vlákien (OpenMP), alebo na úrovni procesov (MPI). Keďže na koprocesore beží vlastný operačný systém, dokáže sa správať ako samostatný systém, aj keď v skutočnosti bez asistencie procesoru fungovať nemôže. Podobne ako karty GPU je k hosťovskému systému pripojený prostredníctvom zbernice PCIe. Medzi hosťovským systémom a koprocesorom je vytvorené vituálne TCP spojenie prostredníctvom PCIe zbernice, čo umožňuje pristupovať ku koprocesoru ako ku sieťovému uzlu. Na jeden hostiteľský systém môže byť pripojených viacero koprocesorov, ktoré môžu medzi sebou komunikovať autonómne, teda bez zásahu procesora.

Koprocesor Xeon Phi disponuje viac ako 50 jadrami (závisý od konkrétneho modelu, pre túto prácu bol použitý model s 60 jadrami), ktoré spolu dodávajú jednotke vysoký výkon. Samotné jadrá využívajú „in-order pipeline“, pričom každé jadro poskytuje 4 hardvérové vlákna (hyperthreading). Pre porovnanie Intel Xeon disponuje 2 hardvérovými vláknami na jadro, pričom ale nevyužíva hyperthreading. Výkon koprocesoru Xeon Phi a procesoru Xeon v závislosti na počte vlákien zobrazuje Obrázok (2). Xeon Phi disponuje 8GB DDR5 pamäťou, ktorá ale pre viac ako 200 vlákien nemusí byť dostatočná. Novšie modely ponúkajú pamäť o veľkosti 16GB.

2.2.1 VPU – jednotka spracovávajúca vektory

Táto jednotka je veľmi dôležitou súčasťou každého jadra, bez nej by bol výkon koprocesoru veľmi nízky. VPU jednotka využíva novú inštrukčnú sadu AVX-512. Jedná sa teda až o 512 bitové SIMD inštrukcie, ktoré v jeden okamih dokážu vykonávať až 16 oprácií s jednoduchou presnosťou (`Float`), ba dokonca pri použití FMA intrukcií až 32 operácií s číslami typu `Float`. Taktiež boli vylepšené „Gather“ a „Scatter“ inštrukcie, ktoré umožňujú nejednotný prístup do pamäti.



Obrázok 2. Porovnanie výkonu Intel Xeon a Intel Xeon Phi na základe počtu vlákien [1].

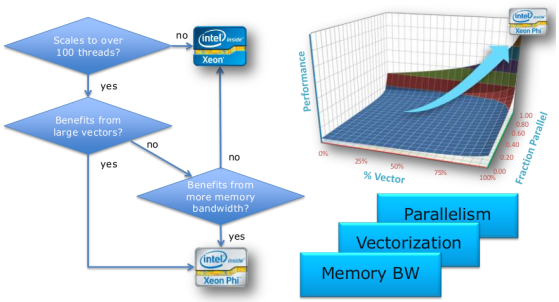
2.2.2 Pamäte Cache

Veľké úsilie bolo venované práve problematike pamätí cache. Pamäťový subsystém koprocesoru ponúka slušnú kapacitu a vysokú priepustnosť. Každé jadro je vybavené 32KB L1 inštrukčnou cache, 32KB L1 dátovou cache a 512KB L2 cache. L1 cache poskytuje asi 15 krát vyššiu priepustnosť pamäte ako hlavná pamať, L2 7 krát. Práve z tohoto dôvodu je efektívne využitie pamätí cache jedným z kľučovým faktorom pri dosahovaní špičkového výkonu.

2.2.3 Programovacie režimy

Intel Xeon Phi ponúka dva programovacie prístupy – „Natívny režim“ a „Offload režim“. **Natívny režim** znamená, že kompilátor generuje kód, ktorý bude spúšťaný výhradne na koprocesore. Je teda naprv nutné preložiť zdrojový kód na hosťovskom systéme, nahrať spustiteľný súbor do pamäte koprocesora, prihlásiť sa na samotný koprocesor a spustiť program. Výhody tohto módu sú rýchlosť a jednoduchosť. Nieje ale vhodný napríklad v prípadoch, keď naša aplikácia vykonáva veľké množstvo I/O operácií, prípadne obsahuje náročné sekcie, ktoré bežia na jednom vlákne. Celkovo nieje vhodné používať tento koprocesor pre aplikácie, ktoré nedokážu využiť aspoň polovicu dostupných vlákien. Všeobecné základné princípy pre voľbu platformy zobrazuje (Obrázok 3)

Offload režim je podobnejší programovaniu grafických kariet, pretože program sa vždy spúšťa na hostiteľskom systéme, pričom niektoré úseky programu môžu byť vykonávané na koprocesore. Je vhodný najmä ak naša aplikácia vykonáva veľké množstvo I/O operácií a využíva malé množstvo vlákien, pričom v programe existujú úseky kódu, ktoré je možné ška-



Obrázok 3. Postup pri voľbe platformy [2].

lovať na viac ako 100 vlákien. Práve tieto úseky kódu sa označia pomocou špeciálnych direktív, kompilátor tým pádom vygeneruje kód ako pre proceror, tak aj pre koprocesor. Ak je po spustení kódu možné vykonať offload, potrebné dáta sa presunú na koprocesor, vykonajú sa operácie a výsledné dáta sa prípadne skopírujú naspäť na procesor. S týmto režimom ale prichádza réžia, ktorú má na svedomí presun dát na/s koprocesora a pod., čo vedie k istému spomaleniu programu.

3. Od teórie k praxi

Teraz, keď už máme približný obraz o architektúre koprocesoru môžme prejsť k praktickejšej časti práce. Dá sa povedať, že táto práca sa z veľkej časti zaoberala štúdiom postupu pri implementácii výpočtovo náročných algoritmov na koprocesore Intel Xeon Phi. V rozsahu tohto dokumentu bohužiaľ nieje možné venovať čas práve týmto postupom, preto si radšej povieme niečo o konkrétnych problémoch, ktoré sme riešili.

3.1 Načo si treba dať pozor

Postup implementácie programov pre Xeon Phi je veľmi podobný postupu pre procesory Xeon. Odporúčaný postup je teda optimalizovať aplikáciu najskôr pre procesor Xeon, potom sa presunúť na koprocesor Xeon Phi.

Samozrejme nieje zaručené, že optimalizovaná procesorová verzia programu pobeží optimálne aj na Xeon Phi (zvyčajne to tak nebýva ...). Tento jav je zapríčinený odlišnosťami v architektúrach. Procesory Xeon používajú najmodernejšie technológie, ktoré sa sústreďujú na špičkový výkon každého jedného jadra. Používajú výkonné VPU jednotky, spracovanie inštrukcií „Out-of-order“, nepoužívajú hyperthreading, disponujú malým počtom vlákien (v porovní s Xeon Phi).

Architektúra Xeon Phi naopak používa podstatne slabšie jadrá (1 vlákno Xeon Phi je asi 40x pomalšie ako 1 vlákno procesoru Xeon !). Túto nevýhodu však kompenzuje veľkým počtom jadier (približne 60), pričom používa hyperthreading. Základom pre dosiahnutie vysokého výkonu je teda použitie **veľkého množstva vlákien**, minimálne 2 na každé jadro. Pre tento účel nám výborne poslúži knižnica `omp.h`, ktorá umožňuje jednoduchú prácu s hardvérovými vláknami (`#pragma omp parallel`, `#pragma omp for ...`).

Veľmi dôležitým krokom k optimalizácii programu je **vektorizácia** výpočtovo náročných úsekov kódu. Xeon Phi dokáže pomocou inštrukcií AVX-512 spracovávať vektory veľmi efektívne. Komplilátor dokáže vektorizovať kód automaticky, niekedy mu však musíme pomôcť direktívami `#pragma ivdep`, `#pragma simd` atď. K dosiahnutiu maximálneho výkonu nám pomôže využívanie inštrukcí FMA, resp. MAD ktoré umožňujú násobenie a sčitovanie operandov v jeden okamih.

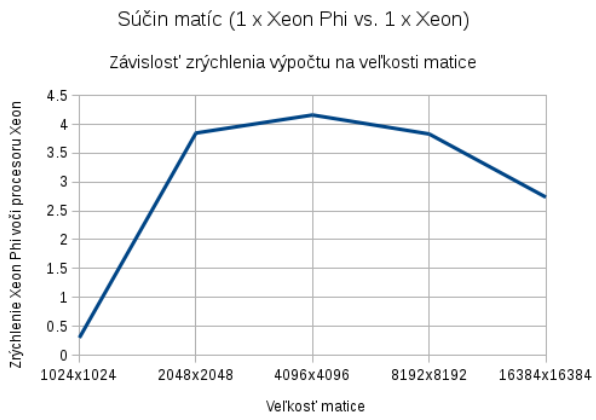
Pre efektívnu prácu vektorových inštrukcií je nutné správne uloženie dát v pamäti. Pre Xeon Phi je ideálne **zarovnanie dát na 64B**. Pre alokovanie pamäte pre zarovnané dáta je možné použiť napríklad funkciu `_mm_malloc()`. Podmienkou vektorizácie je taktiež nezávislosť spracovávaných vektorov, resp. vektory sa nesmú v pamäti prekrývať („pointer aliasing“).

Všetky experimenty spojené s programovaním a využívaním koprocesoru Xeon Phi boli uskutočňované na Ostravskom superpočítači Anselm. Anselm ponúka 4 uzli obsahujúce Xeon Phi. Každý uzol okrem koprocesora obsahuje 2 krát procesor Intel Xeon. Teoretický výkon týchto dvoch procesorov je asi 1/3 výkonu Xeon Phi.

3.2 Jednoduché benchmarky

Práca s Xeon Phi bola pre mňa nová, preto bolo najskôr nutné začať s programami, na ktorých by sa dala jednoducho vyskúšať práca s naším koprocesorom. Začalo sa teda s benchmarkmi typu **Súčin matice a vektora**, **Súčin matíc** a pod., ktoré načrtli kde približne sa skrývajú problémy, ktoré je potrebné riešiť. Začali sme vlastnou implementáciou, ktorou sme sa snažili výkonovo priblížiť funkcii s knižnice MKL. Pre porovnanie výkonu procesoru Xeon a koprocesoru Xeon Phi pri násobení matíc bola zvolená funkcia `cblas_dgemm`.

Výsledky nás milo prekvapili. Pri násobení 2 matíc napríklad o veľkosti 512MB bol Xeon Phi 3,5 krát rýchlejší ako 1 procesor Intel Xeon. Xeon Phi bol v porovnaní s 2 procesormi Xeon rýchlejší 1,8 krát (512MB matice). Výsledky sa javia ako celkom slušné, najmä ak zvážime pomer cena/výkon (cena 2 procesorov Xeon je podstatne vyššia ako 1 Xeon Phi). Priebeh zrýchlenia výpočtu na Xeon Phi v závislosti na



Obrázok 4. Zrýchlenie Xeon Phi voči procesoru Xeon pri násobení matíc

veľkosti matice zobrazuje (Obrázok 4).Viac výsledkov meraní, ukážky zdrojových kódov a grafy zrýchlenia voči procesoru je možné nájsť na odkaze v priložených materiáloch, adresár `matmul`.

3.3 N-body

Jedná sa o komplexnejší problém, ktorý zahŕňa podstatne viac výpočtov ako predchádzajúce príklady. Ide o sériu výpočtov síl pôsobiacich medzi jednotlivými časticami (Rovnica 1), zrýchlenia častíc (Rovnica 2), rýchlosti častíc (Rovnica 3) a polohy častíc (Rovnica 4). Aplikácia simuluje pohyb častíc v priestore, na základe síl pôsobiachich medzi nimi. Riadime sa teda nasledujúcimi, dobre známymi vzorcami.

$$F = \frac{G * m1 * m2}{R^2} \tag{1}$$

$$a^{(i+1)} = \frac{\sum F_n^{i+1}}{m} \tag{2}$$

$$v^{(i+1)} = v^i + a^{(i+1)} * \Delta T \tag{3}$$

$$r^{(i+1)} = r^i + v^{(i+1)} * \Delta T \tag{4}$$

Predchádzajúce rovnice bolo teda najskôr nutné prepísať do jazyka C++, otestovať správnosť výpočtu a optimalizovať. Pri optimalizácii aplikácie pre Xeon Phi je potrebné vektorizovať kritické úseky kódu a škálovať ich aspoň na polovicu dostupných vlákien (v našom prípade 120 až 240). Ďalšou dôležitov súčasťou optimalizácie bolo zajistenie efektívneho prístupu do pamäte, resp. čo najlepšie využitie pamätí cache. Vzniklo teda viacero verzií aplikácie, ktoré boli medzi sebou porovnávané. Simulácia bola spočiatku vyvíjaná pre procesor Intel Xeon, neskôr bola presunutá a optimalizované pre koprocesor Intel Xeon Phi.

Po dostatočnej optimalizácii aplikácii prišlo narad experimentovanie z rôznymi verziami programu a ich porovnávanie medzi procesorom a koprocesorom.Pri behu aplikácie na 1 procesore Intel Xeon (8 vlákien) bol dosiahnutý výkon takmer 200 gigaFLOPS (2x Intel Xeon 390 gigaFLOPS), pre čísla typu `float`. Koprocesor Intel Xeon Phi (240 vlákien) dokázal pre tento problém vyprodukovať takmer 600 gigaFLOPS. Tento algoritmus ani z ďaleka nepracuje s takým objemom dát (simulácia prebehla napríklad pre 100000 častíc, cca 5MB pamäti) ako predchádzajúci.

Veľkosť spracovávaných dát je teda ďalšou kľúčovou vlastnosťou pri voľbe platformy. Xeon Phi potrebuje pre optimálne využitie svojich 240 vlákien omnoho viac Pamäte ako procesor s 8 vláknami (tento fakt je samozrejme závislí aj na type algoritmu). Viac výsledkov meraní, ukážky zdrojových kódov a graf škálovania je možné nájsť na odkaze v priložených materiáloch, adresár `nbody`.

3.4 k-Wave

Veľmi zaujímavou súšasťou tejto práce, bola snaha o využitie Xeon Phi pre modul MATLABu **k-Wave** [3]. Modul k-Wave je určený pre simuláciu šírenia akustických vĺn v 1D, 2D alebo 3D. Plán bol teda taký, že sa vezme optimalizovaná C++ verzia k-Wave využívajúca OpenMP a prenesie sa na koprocesor Intel Xeon Phi. Kompilácia aplikácie pre Xeon Phi však nebola jednoduchá, pretože využívala neštandartné knihovne, ktoré bolo nutné prispôsobiť potrebám Xeon Phi.

Po úspešnej kompilácii ale prišlo sklamanie, pretože program bežal na Xeon Phi asi 2 krát pomalšie ako na procesore. Aplikácia pre drvivú časť výpočtov využíva matematické funkcie (FFT) z knižnice Intel MKL. Po profilovaní aplikácie sa zistilo, že problematické sú práve funkcie z knižnice MKL, ktoré sú z neznámych dôvodov na koprocesore pomalšie ako na procesore.

Pokúsili sme sa teda zmerať výkon FFT funkcií na procesore pomocou jednoduchých benchmarkov. 1 procesor Xeon pri tomto probléme produkoval výkon približne 70 gigaFLOPS (v závislosti od veľkosti spracovávaných dát a pod.). 2 procesory Xeon produkovali výkon 115 gigaFLOPS. Xeon Phi bol vo väčšine prípadov pomalší, alebo sa iba ťažko dotiahoval na výkon procesorov. Okrem uskutočnených meraní môžme na webe nájsť aj iné výsledky, kde sa výkon Xeon Phi pri FFT pohyboval taktiež okolo 100 gigaFLOPS. Tento problém sa nám bohužiaľ doposial nepodarilo vyriešiť, možno s novou verziou MKL príde aj vyšší výkon Xeon Phi pre FFT funkcie.

3.5 Kroskompilácia existujúcich knižníc, modulov, programov

Veľmi zaujímavou časťou práce bola snaha kroskompilácie rôznych existujúcich knižníc, modulov a pod. Jednalo sa napríklad o HDF5, ktorá je datovým modelom, knižnicou a formátom súborov pre ukladanie a spravovanie dát. Ďalej to bola napríklad knižnica ZLIB pre kompresiu dát, či dokonca interpret čoraz populárnejšieho jazyka Python a jeho výnamných modulov Numpy a Scipy. Moduly Numpy a Scipy obsahujú veľké množstvo optimalizovaných funkcií, metód či dátových typov, ktoré sa využívajú pri náročných matematických, fyzikálnych a štatistických výpočtoch. Podarilo sa nám, aby vyššie spomenuté veci bežali natívne, priamo na koprocesore Xeon Phi.

3.6 Extrakcia ivectoru

Python bežiaci natívne na karte Xeon Phi bol určený pre výzkumnu skupinu FIT VUT, ktorá sa zaoberá spracovaním hovorenej reči. Jednalo sa o takzvanú extrakciu ivectoru, ktorá tvorí veľmi významnú časť procesu spracovania reči. Aplikácia bola vytvorená v jazyku Python, pričom využívala volania funkcií z Intel MKL.

Po kompilácii interpretu Python pre Xeon Phi nás koprocesor znova prekvapil, tento krát nie príliš pozitívne. Funkcie z knižnice Intel MKL skutočne vykazovali zrýchlenie oproti procesoru, avšak úseky kódu v pythone vykonávané sekvenčne celý program pochovali. Ako už bolo spomenuté, Xeon Phi má nie príliš výkonné jadrá, najmä ak sa kód vykonáva len na jednom vlákne. Pre porovanie, doba behu aplikácie na Xeon Phi bola asi 20 krát dlhšia ako na procesore.

4. Záver

[Súhrn] Tento dokument popisuje prácu s Xeon Phi len veľmi stručne a abstraktnejšie, nezaoberá sa napríklad množstvom detailov ohľadom implementácie výpočtovo náročných algoritmov na Xeon Phi, ktoré sú rozoberané v bakalárskej práci. Čitateľ by ale mal byť oboznámený o základnom cieli práce, o dosiahnutých výsledkoch, atď. Touto cestou by som sa chcel veľmi pekne poďakovať vedúcemu mojej bakalárskej práce Ing. Jiřímu Jarošovi Ph.D., ktorý mi počas tvorby práce poskytol cenné rady a skúsenosti.

[Dosiahnuté výsledky] Dosiahnuté výsledky boli priebežne obsiahnuté v predchádzajúcich kapitolách, najmä 3.3, 3.4 a 3.5. Okrem toho bol na základe získaných vedomostí a skúseností zhodnotený potenciál koprocesoru, ktorý si zosumarizujeme v nasledujúcom odseku – Záverečná úvaha.

[Záverečná úvaha] Z množstva vykonaných benchmarkov a pri riešení zložitejších úloh sme si vytvorili obraz o možnostiach využitia koprocesoru. Xeon Phi má zmysel využívať pre vysoko optimalizované paralélne úlohy. Je veľmi dôležité využívať paralélne spracovanie ako na úrovni vlákien, tak aj na úrovni inštrukcií SIMD. Hodí sa napríklad aj pre algorimty, ktoré okrem náročných výpočtov spracovávajú veľké množstvo dát (samozrejme paralélne).

Xeon Phi môže napríklad v porovnaní s najnovšími akcelerátormi GPGPU, ktoré dosahujú až niekoľko násobne vyšší výkon výrazne zaostávať. Xeon Phi ale môže mať výhodu pri úlohách, ktoré sú pre extrémne jednoduché jadrá GPGPU príliž zložité. Jadrá koprocesoru (i keď sú pomalé) totiž vznikli s jadier procera, ktoré dokážu riešiť omnoho viac problémov ako GPGPU. Okrem toho je možné využívať koprocesor ako samostaný systém, ktorý aj pri plnom vyťažení beží autonómne, bez zásahu hostiteľského systému.

Ako už bolo spomenuté, koprocesor má v porovnaní s procesorom Xeon naviac pri vysoko optimalizovaných a paralélnych úlohách. Je možné použiť offload režim, kedy program beží primárne na procesore a niektoré časti výpočtu sú presunuté na koprocesor. Okrem toho môžu na riešení problému procesor a koprocesor pracovať paralélne. Rozhodne nieje vhodné spúšťať na koprocesore úlohy, ktoré obsahujú náročné úseky kódu spracovávané sekvenčne. Pripomeniem, že 1 vlákno Xeon Phi je asi 40 krát pomalšie ako 1 vlákno procesoru Xeon.

[Plány do budúcna] V budúcnosti by sme sa chceli zamerať napríklad na využitie viacerých koprocesorov súčastne (napr. pomocou MPI) pri riešení zložitých úloh, portovaním ďalších existujúcich programov, modulov, knižníc na Xeon Phi apod.

Literatúra

[1] *Intel Xeon Phi Coprocesor High Performance Programming*. Elsevier Inc.y, 2013. ISBN: 978-0-12-410414-3.

[2] Intel. Intel xeon phi coprocessor – the architecture, 2012. https://software.intel.com/en-us/articles/intel-xeon-phi-coprocessor\-codename-knights-corner.

[3] B. E. Treeby, J. Jaros, A. P. Rendell, and B. T. Cox. Modeling nonlinear ultrasound propagation in heterogeneous media with power law absorption using a k-space pseudospectral method, 2012. http://www.k-wave.org/papers/2012-Treeby-JASA.pdf.