Semestrální projekt MI-PAP 2015/2016:

Modelování částic

Miroslav Brabenec
Jan Nováček
magisterské studium, FIT ČVUT, Thákurova 9, 160 00 Praha 6

14. května 2016

1 Definice problému, popis sekvenčního algoritmu a jeho implementace

1.1 Definice problému

Implementujte tento algoritmus (viz `http://www.browndeertechnology.com/docs/BDT_OpenCL_Tutorial_NBody-rev3.html#algorithm`) a upravte ho takto:

1. částice při nárazu provedou pružný odraz a
2. každá částice má svůj náboj

Úkol: doplňte o možnost vizualizace (alespoň exportem dat v daných časových okamžicích ve formátu vhodném pro vizualizaci v nástroji třetích stran např.)

1.2 Popis sekvenčního algoritmu

Simulátor načte nastavení simulace ze vstupního souboru, z toho nainicializuje hodnoty a spustí simulaci.

V každém kroku simulace se vypočítává nová pozice každé částice, nová pozice záleží na vlastnostech dané částice a také na vlivu ostatních částic na tuto částici.

Simulace končí po provedení zadaného počtu kroků.

Vizualizovat simulaci lze dvěma způsoby. První umožňuje sledovat průběh simulace jako animaci přímo za běhu, k tomu byla využita knihovna CImg.

Druhý způsob využívá možností programu gnuplot, simulátor vytvoří zdrojový soubor pro vizualizaci v gnuplotu a zároveň vytvoří soubor se zdrojovými daty, která se mají vykreslit.

1.3 Sekvenční implementace

Bylo implementováno několik sekvenčních variant, lišících se v míře ruční vektorizace struktur použitých v algoritmu.

První varianta (V1) používá pouze automatickou optimalizaci, nepoužívá žádné SSE instrukce ani ručně vytvořené vektory.

V druhé variantě (V2) je automatická optimalizace a vektorizace.

Ve třetí variantě (V3) je automatická optimalizace a vektorizace, SSE datové typy __m128 a SSE instrukce pro výpočet $\frac{1}{\sqrt{x}}$.

Porovnání rychlostí jednotlivých sekvenčních implementací je v tabulce níže.

Počet částic	**Počet kroků sim.**	**V1**	**V2**	**V3**
1024	50 000	760.19s	149.38s	97.98s
512	400 000	1526.06s	357.50s	202.17s
4096	5 000	1215.77s	207.57s	153.81s
8192	1 500	1459.20s	242.25s	184.60s

Tabulka 1: Časy sekvenčních implementací

2 Popis paralelního algoritmu a jeho implementace v OpenMP

Paralelizace algoritmu proběhla tak, že se v každém kroku simulace rozdělí částice mezi zadaný počet vláken, paralelně se tedy počítají nové pozice částic. Rozdělení částic mezi vlákna bylo ponecháno na OpenMP.

2.1 Vektorizace

V programu byla nejprve použita automatická vektorizace (-O3 -mavx). Její výsledek měl na rychlost výpočtu minimální vliv. Výrazné zrychlení nastalo, až po přidání přepínače -ffast-math. Díky tomuto přepínači mohl kompilátor použít vektorové instrukce pro výpočet $\frac{1}{\sqrt{x}}$.

V další fázi byl kód ručně upraven, tak aby bylo možné využít vektorové instrukce, které dokáží počítat paralelně několik hodnot $\frac{1}{\sqrt{x}}$. Zde byly částice rozděleny do chunků po 4 a v každém kroku se počítá ovlivnění částice chunkem částic. Tato varianta vykazovala další zrychlení cca o 33% oproti předchozí variantě (-O3 -mavx -ffast-math).

2.2 Optimalizace

Pro ruční vektorizaci byly využity technologie obsažené již v SSE2 (datový typ __m128). Dalšího zrychlení by bylo možno dosáhnout při použití novějších vektorových instrukcí. Ty by měly umožňovat tvorbu chunků o velikosti 8, případně 16. Dále by také bylo možno upravit kód, tak aby docházelo k počítání vzdálenosti mezi částicemi pouze jednou. Momentálně se počítá vzdálenost mezi částicemi A a B dvakrát, jednou z pohledu částice A a jednou z pohledu částice B.

2.3 Vstupní data

Pro měření jsme vytvořili 4 vzorky dat. Vzorky dat se lišily v počtu částic a počtu kroků simulace. Všechny vzorky byly srovnatelné co se týče výpočetní náročnosti. Výpočetní náročnost lze spočítat jako $N = P^2 \times S$, kde P = počet částic a S = počet kroků. Efektivita paralelního výpočtu se zlepšuje s rostoucím počtem částic.

Počet částic	**Počet kroků sim.**	**Výpočetní náročnost**[1]
1024	50 000	52,428
512	400 000	104,857
4096	5 000	83,804
8192	1 500	100,663

Tabulka 2: Vzorky použité pro srovnání sekvenčních implementací

Po naměření časů sekvenčních implementací bez vektorizace a s vektorizací jsme usoudili, že pro měření paralelních implementací zvýšíme výpočetní náročnost úloh. Samotná vektorizace zrychlila sekvenční část mnohonásobně - simulace běžící sekvenčně bez vektorizace téměř 25 minut trvala s vektorizací 3-4 minuty.

[1]Spočítáno podle vzorce $\frac{P^2 \times S}{10^9}$, kde P = počet částic a S = počet kroků

Měření	Počet částic	Počet kroků sim.	Výpočetní náročnost
1	1024	150 000	157,284
2	512	1 200 000	314,571
3	4096	15 000	251,412
4	8192	4 500	301.989

Tabulka 3: Nové zadání pro měření paralelního zrychlení

3 Naměřené výsledky a vyhodnocení pro OpenMP

K měření výkonu paralelní implementace bylo původní zadání pozměněno tak, aby bylo výpočetně náročnější, viz předchozí tabulka. U každého měření je uvedena tabulka s počtem vláken a časem výpočtu ve vteřinách, informace z tabulky jsou dále vyneseny do grafu. Na konci měření k jednotlivým variantám jsou grafy s paralelním zrychlením. Na konci sekce je porovnání a vyhodnocení měření.

3.1 Měření pro variantu 2

Varianta 2 využívá automatické vektorizace.

Počet vláken	Naměřený čas
1	448.140625s
2	422.681641s
4	269.634277s
8	253.593750s
12	220.276367s
16	188.523438s
24	188.625000s

Tabulka 4: Časy 1. měření

Počet vláken	Naměřený čas
1	1066.172363s
2	1004.078125s
4	773.147461s
8	765.203125s
12	807.484375s
16	782.453125s
24	640.501953s

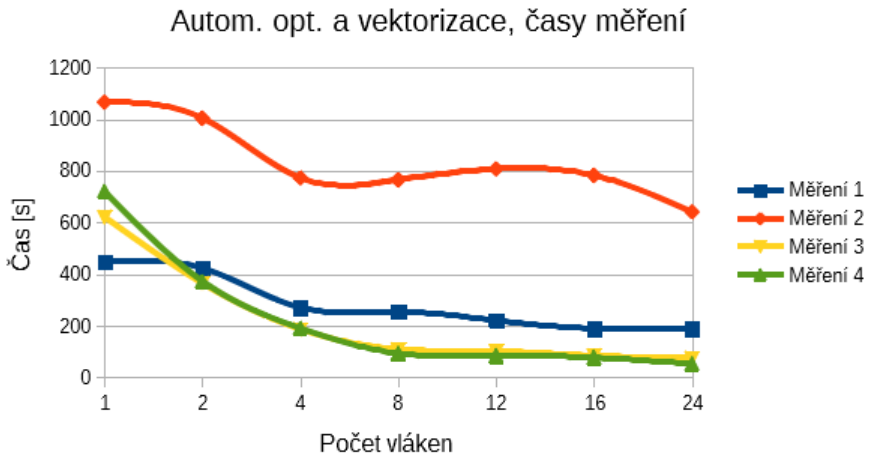
Tabulka 5: Časy 2. měření

Počet vláken	Naměřený čas
1	621.765625s
2	364.220703s
4	185.841797s
8	109.250000s
12	101.171875s
16	84.234375s
24	74.109375s

Tabulka 6: Časy 3. měření

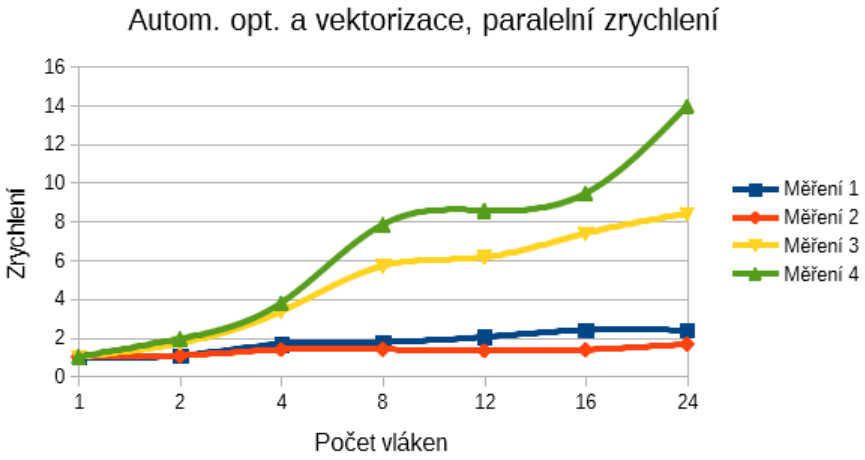
Počet vláken	Naměřený čas
1	720.630859s
2	372.531250s
4	190.187500s
8	92.015625s
12	84.015625s
16	76.273438s
24	51.625000s

Tabulka 7: Časy 4. měření



Obrázek 1: Výsledky měření

Z naměřených výsledků lze usoudit, že paralelizace algoritmu zredukuje výpočetní čas znatelně, až 14×. Z měření je vidět, že čím více částic bylo v simulaci, tím vyššího zrychlení bylo dosaženo. Vyšší počet kroků simulace naopak zrychlení spíš srážel, někdy dokonce zpomaloval. Lineárního zrychlení dosaženo nebylo, jak je vidět z následujících grafů.



Obrázek 2: Paralelní zrychlení

3.2 Měření pro variantu 3

Ve třetí variantě (V3) je automatická optimalizace, automatická a ruční vektorizace, mj. SSE datové typy __m128 a SSE instrukce pro výpočet $\frac{1}{\sqrt{x}}$.

Počet vláken	Naměřený čas
1	292.988647s
2	321.015625s
4	206.636963s
8	191.578125s
12	189.591797s
16	159.812500s
24	144.281250s

Tabulka 8: Časy 1. měření

Počet vláken	Naměřený čas
1	607.515625s
2	675.796875s
4	592.843750s
8	683.521484s
12	845.265625s
16	696.267578s
24	587.625000s

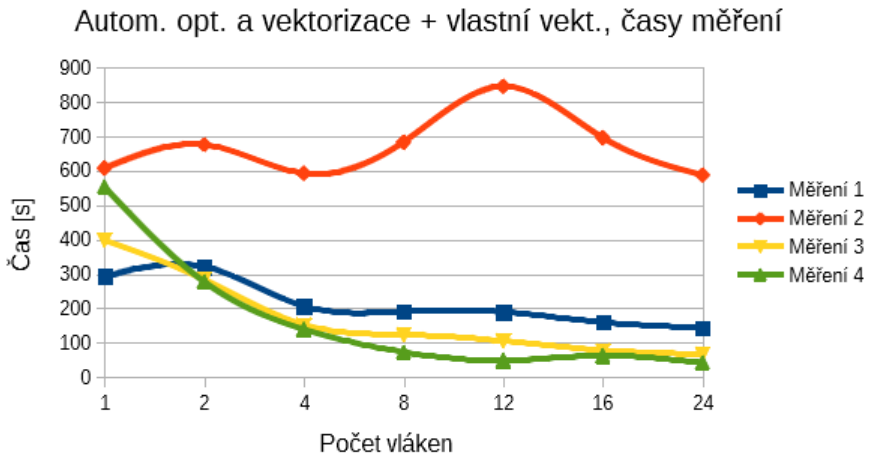
Tabulka 9: Časy 2. měření

Počet vláken	Naměřený čas
1	398.190857s
2	284.056641s
4	152.609375s
8	123.656250s
12	105.606934s
16	78.484375s
24	67.243164s

Tabulka 10: Časy 3. měření

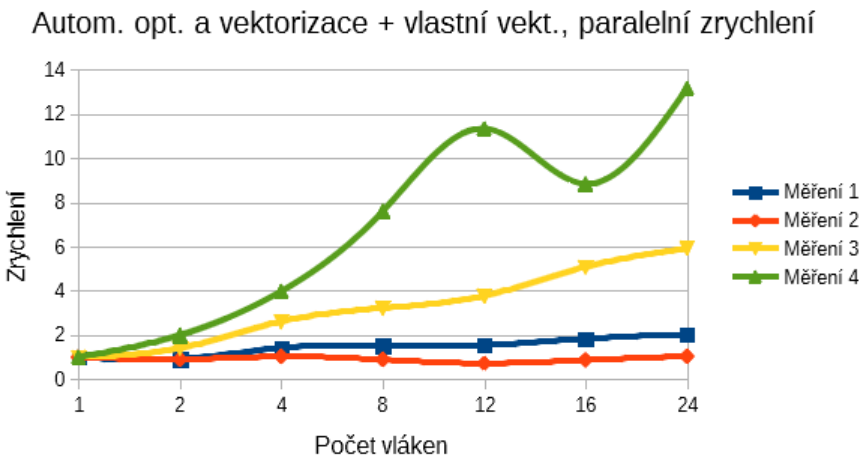
Počet vláken	Naměřený čas
1	553.531250s
2	277.496094s
4	139.218750s
8	72.937500s
12	48.828125s
16	62.685547s
24	42.093750s

Tabulka 11: Časy 4. měření



Obrázek 3: Výsledky měření

Paralelizace varianty č. 3 zrychlila výpočet až 13×. Naměřené výsledky ukazují, že efektivnější využití více vláken ve variantě č. 3 se projeví zejména při vyšším počtu částic. Následují grafy paralelního zrychlení.



Obrázek 4: Paralelní zrychlení

3.3 Porovnání jednotlivých implementací

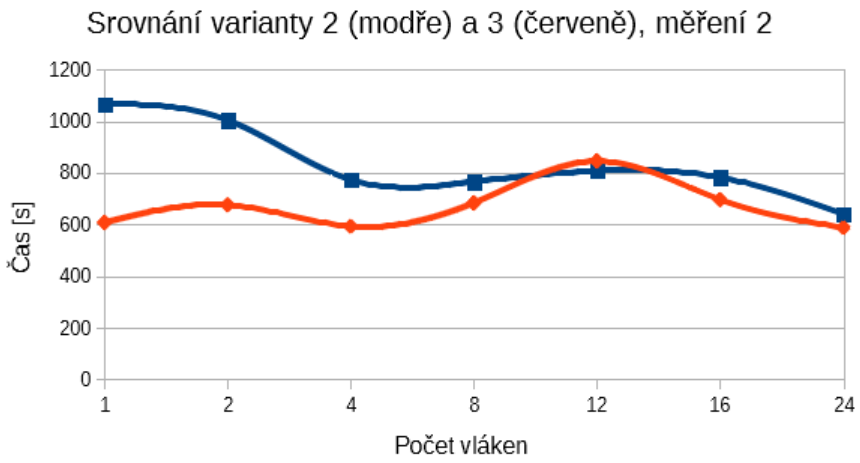
První varianta algoritmu, tj. sekvenční s pouze automatickou vektorizací (-O3 -mavx), vyšla jako nejpomalejší ze všech variant.

Vektorizace v dalších dvou variantách přinesla obrovské zrychlení, varianta 2 je oproti variantě 1 až 6× rychlejší, varianta 3 oproti variantě 1 přibližně 8×.

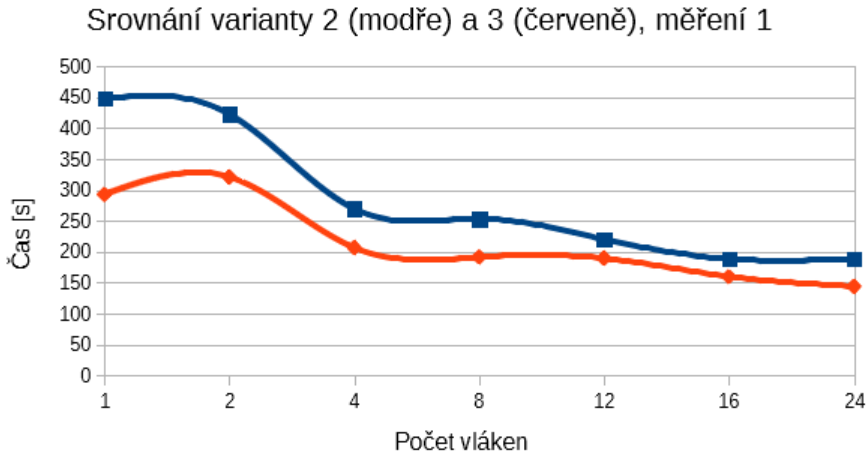
Spuštění varianty 2 na více vláknech přineslo zrychlení až 14×. Z jednotlivých měření lze usoudit, že zrychlení se zvyšuje i s větším množstvím částic. Varianta 2 nikdy výpočet nezpomalila, její sekvenční implementace sice je pomalejší než varianta 3, ale v paralelizaci dosahovala většího zrychlení.

Spuštění varianty 3 na více vláknech zrychlilo výpočet až 13,1×. Bohužel to v některých případech výpočet i zpomalilo, to přisuzujeme příliš malému množství částic v kombinaci s režií spojenou s vlákny. Zrychlení je naopak patrné v měření 3 a 4, kdy se při větším počtu částic výpočet znatelně zrychloval s přibývajícími vlákny. Varianta 3 také těžila z velmi dobrého sekvenčního času, který předčil i variantu 2 s automatickou opt. a vektorizací.

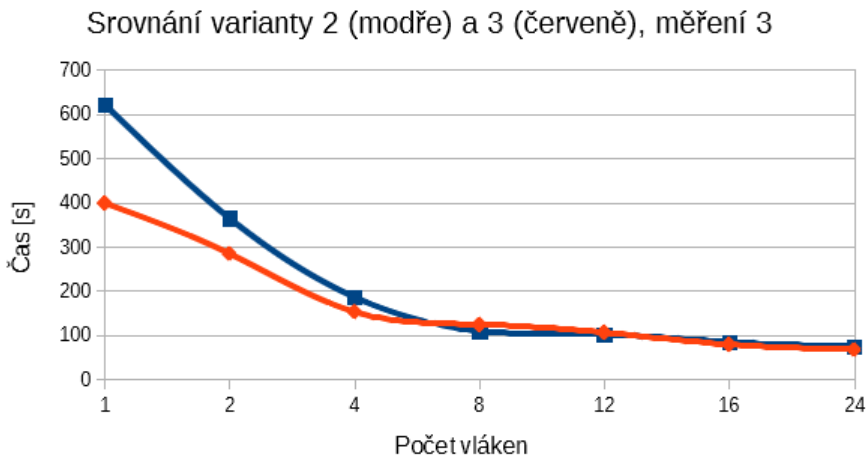
Dále jsou uvedeny grafy porovnávající variantu 2 a 3, ukazují vliv vektorizace na rychlost výpočtu.



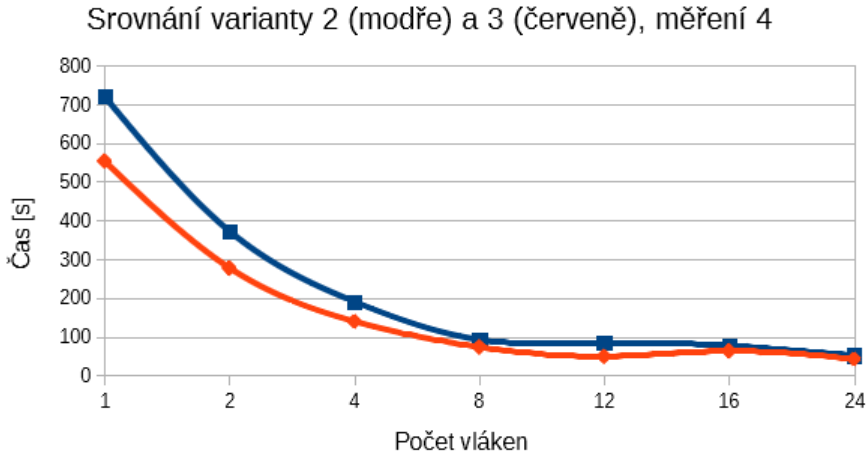
Obrázek 6: Porovnání implementací, měření 2



Obrázek 5: Porovnání implementací, měření 1



Obrázek 7: Porovnání implementací, měření 3



Obrázek 8: Porovnání implementací, měření 4

4 Naměřené výsledky a vyhodnocení pro Xeon Phi

Program pro Xeon Phi byl vytvořen z původního zdrojového kódu jeho kompilací přes icc, tj. kompilátor od Intelu. Výsledný program je tedy podobný variantě 2 z předchozí sekce, jen jsou automatické optimalizace a vektorizace v režii jiného kompilátoru a přizpůsobené jiné architektuře.

4.1 Co se měřilo

Měření proběhlo na stejných datech, pro připomenutí jsou jednotlivá zadání uvedena v následující tabulce.

Měření	**Počet částic**	**Počet kroků sim.**	**Výpočetní náročnost**
1	1024	150 000	157,284
2	512	1 200 000	314,571
3	4096	15 000	251,412
4	8192	4 500	301.989

Tabulka 12: Zadání měření - pro připomenutí

4.2 Vektorizace

Kompilátor icc provedl automatickou vektorizaci, při které došlo k vektorizaci stejného bloku kódu jako u kompilátoru g++. Vektorizován byl vnitřní cyklus kódu, který počítá to jak se částice navzájem ovlivňují. Vektorizace tohoto úseku kódu má velký vliv na výkonost. Zrychlení je způsobeno tím, že v tomto úseku kódu je třeba počítat vzdálenost 2 částic. V tomto výpočtu figuruje inverzní odmocnina. Díky vektorizaci lze spočítat inverzní odmocninu rychleji a zároveň lze počítat několik hodnot najednou.

4.3 Výsledky měření bez vektorizace

V této sekci jsou uvedeny tabulkově výsledky měření podle počtu vláken pro jednotlivá měření. Měření je graficky znázorněno v předposlední sekci věnované Xeonu Phi, kde je zároveň i grafické porovnání s variantou s vektorizací a grafy paralelního zrychlení.

Počet vláken	Naměřený čas
61	158.770535s
122	109.994492s
244	73.545292s

Tabulka 13: Časy 1. měření

Počet vláken	Naměřený čas
61	433.739839s
122	338.331224s
244	311.221709s

Tabulka 14: Časy 2. měření

Počet vláken	Naměřený čas
61	219.587957s
122	142.293198s
244	80.381402s

Tabulka 15: Časy 3. měření

Počet vláken	Naměřený čas
61	258.860143s
122	167.851305s
244	91.632020s

Tabulka 16: Časy 4. měření

4.4 Výsledky měření s vektorizací

V této sekci jsou výsledky jednotlivých měření s vektorizací. Grafické znázornění je v následující sekci, včetně grafů paralelního zrychlení.

Počet vláken	Naměřený čas
61	98.085072s
122	116.254455s
244	50.155506s

Tabulka 17: Časy 1. měření

Počet vláken	Naměřený čas
61	393.132266s
122	431.340725s
244	272.362233s

Tabulka 18: Časy 2. měření

Počet vláken	Naměřený čas
61	30.253954s
122	18.628104s
244	11.668549s

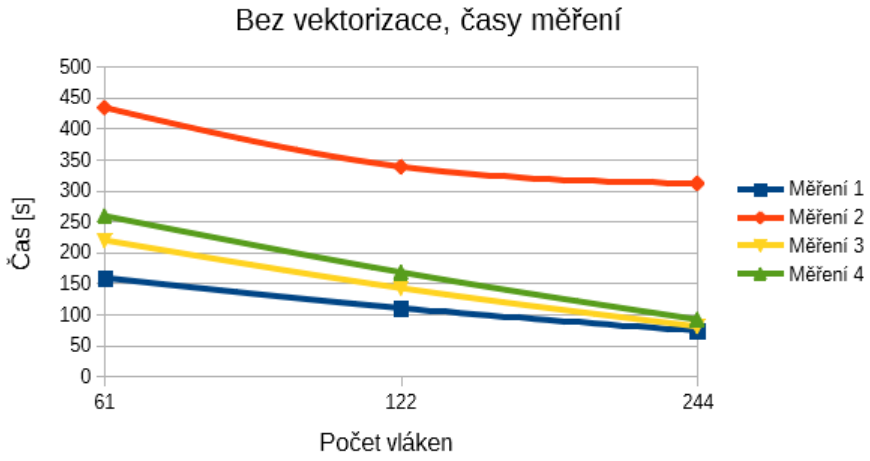
Tabulka 19: Časy 3. měření

Počet vláken	Naměřený čas
61	28.133410s
122	17.696219s
244	11.387652s

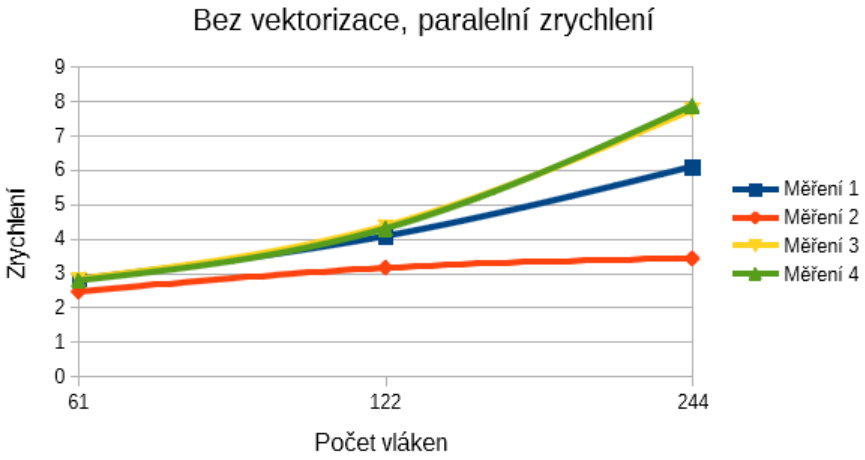
Tabulka 20: Časy 4. měření

4.5 Grafické znázornění výsledků měření a paralelního zrychlení

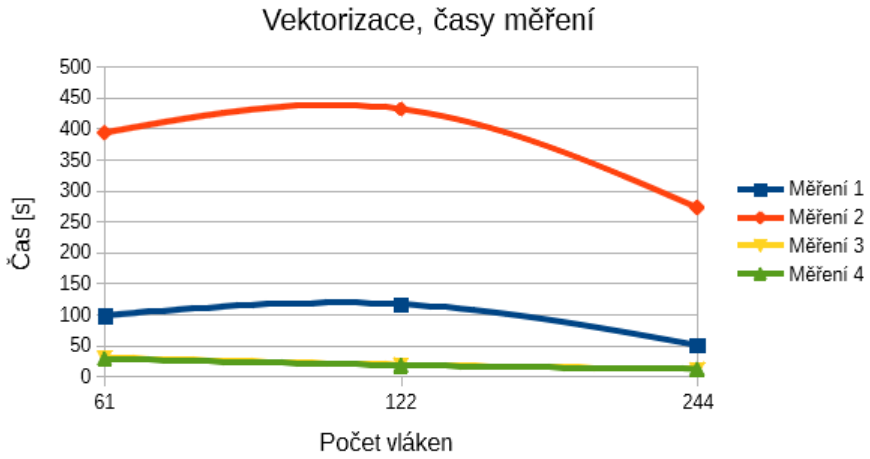
V této sekci jsou do grafů vyneseny časy z jednotlivých měření pro Xeon Phi. Poté následují grafy paralelního zrychlení pro obě varianty. K výpočtu paralelního zrychlení byly použity sekvenční časy varianty 2 z měření pro klasické procesory, což byla verze s automatickými optimalizacemi a vektorizací provedené kompilátorem g++.



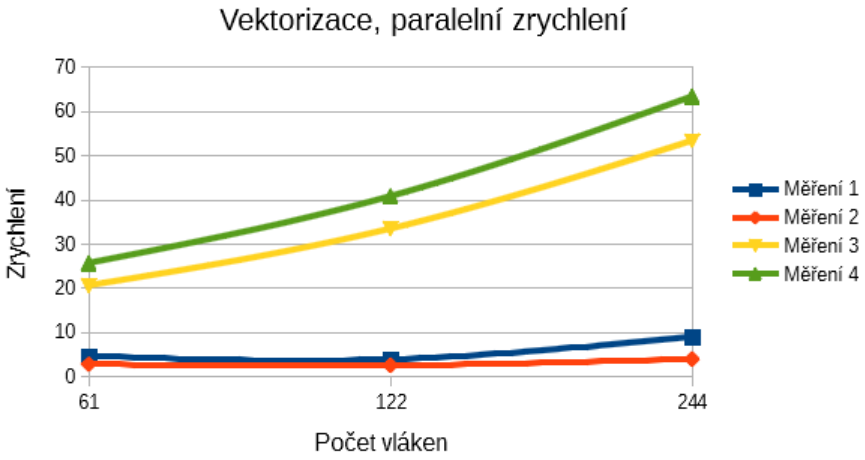
Obrázek 9: Výsledky měření bez vektorizace



Obrázek 10: Paralelní zrychlení bez vektorizace



Obrázek 11: Výsledky měření s vektorizací



Obrázek 12: Paralelní zrychlení s vektorizací

4.6 Vyhodnocení měření

Ve všech měřeních byl výpočet na Xeonu Phi rychlejší než výpočet provedený na klasických procesorech. O kolik rychlejší pak záleželo na vektorizaci. Bez vektorizace dosahovalo zrychlení hodnot až 7,8 v měření č. 4. S vektorizací bylo dosaženo zrychlení až 63 v měření č. 4. Lineárního zrychlení dosaženo nebylo.

Zrychlení v měření 1 a 2 v obou variantách dosahovalo znatelně nižších hodnot, program bez vektorizace byl při některých počtech vláken dokonce rychlejší než varianta s vektorizací.

V měřeních 3 a 4 se naopak hodně ukázala síla vektorizace, kdy zrychlení dosahovalo řádově několika desítek, naproti tomu varianta bez vektorizace pouze v řádu jednotek.

Celkově dopadlo měření podle očekávání. Kompilátor provedl vektorizaci podobným způsobem jako kompilátor pro klasické procesory, algoritmus ovšem dokázal zužitkovat všech 244 vláken Xeonu Phi a čas výpočtu simulace tak byl výrazně nižší.

5 Naměřené výsledky a vyhodnocení pro CUDA

V sekci je popis implementace, výsledky měření a vyhodnocení simulace využívající technologii CUDA, výpočet běží na grafické kartě.

5.1 Popis implementace

V této sekci je popis implementace za použití technologie CUDA.

Zparalelizován byl vnější cyklus výpočtu nových pozic částic tak, aby každé vlákno dostalo svou jednu částici a pro tuto částici vypočetlo zrychlení (vnitřní cyklus simulace). Paralelizace vnitřního cyklu by byla problematická a pravděpodobně by se nevyplatila, důvody jsou velmi krátký výpočet, omezená velikost sdílené paměti multiprocesoru, pomalejší čtení z globální paměti a nutnost paralelní redukce (případně i nad globální paměti) jednotlivých zrychlení, resp. vektorů zrychlení.

Co se týče využití globální paměti, v té jsou uloženy souřadnice částic jako vektor typu float4 (x, y, z, hmotnost) a rychlosti částic jako typ float4 (vx, vy, vz, 0). Důvod volby typu float4 je v čtení všech informací o částici v jednom jediném čtení z globální paměti a čtení z paměti po 128B. Tato úprava vyžadovala vytvořit převodník mezi formátem několika za sebou jdoucích lineárních polích (jednotlivě pro x, y, z, m, atd.) do float4 (x, y, z, m) a (vx, vy, vz, 0).

Sdílená paměť se využívá jako cache pro částice z globální paměti. Každé vlákno ve svém bloku přečte částici z globální paměti na indexu vypočteném z indexu daného vlákna a sub-bloku. Sub-blok značí aktuální podmnožinu částic, jejichž vliv se v aktuální iteraci připočítává k částici vlákna. Dále částici uloží do sdílené paměti na indexu rovném indexu vlákna. Poté, co každé vlákno bloku toto čtení a uložení provede, začne každé vlákno počítat vlivy ostatních částic na svou přidruženou částici, přičemž informace o ostatních částicích bere právě z rychlejší sdílené paměti, nikoliv z globální.

Čtení/zápis z/do globální paměti je tedy nutný v každém jednom vlákně v každém kroku simulace, ale díky využití typu float4 a sdílené paměti by měla být doba přístupu a počet přístupů do globální paměti výrazně nižší.

Vizualizace řešení je stejná jako v předchozích variantách - přes gnuplot (program vytvoří zdrojový kód pro gnuplot, vygenerovat graf je potřeba ručně). Dále je možné vizualizovat přímo za běhu simulace přes knihovnu CImg, kdy je ale celkem velká režie v samotném vykreslování a přenosu dat z GPU na CPU.

5.2 Nastavení kompilátoru

Ke kompilaci byl použit kompilátor g++ a nvcc. Za zmínku stojí využití přepínače -O3, tedy optimalizace. OpenMP bylo přilinkováno kvůli měření času (ten byl měřen jak přes cudaEvent, tak přes omp_get_wtime()).

Příkazy pro kompilaci pro celý program byly následující:

```
g++ ./generator/ioproc.cpp -c -o ioproc.o;
g++ ./generator/SimConfig.cpp -c -o SimConfig.o
nvcc -O3 ./main.cu -c -o main.o -Xcompiler -fopenmp

nvcc -O3 ioproc.o SimConfig.o main.o -o \
  simulator -Xcompiler -fopenmp -lcuda -lcudart -lgomp
```

5.3 Měřené instance

Měření proběhlo na stejných datech jako pro Xeon Phi a vektorizované implementace pro klasické procesory. Pro připomenutí jsou jednotlivá zadání uvedena v následující tabulce.

Měření	Počet částic	Počet kroků sim.	Výpočetní náročnost
1	1024	150 000	157,284
2	512	1 200 000	314,571
3	4096	15 000	251,412
4	8192	4 500	301.989

Tabulka 21: Zadání jednotlivých měření

5.4 Konfigurace bloků a vláken

Byly zvoleny celkem 4 konfigurace bloků a vláken.

První konfigurace měla přiřadit co možná nejvíc vláken každému jednotlivému bloku, počet bloků se poté dopočítal. Počet vláken na 1 blok jsme nastavili na polovinu maximálního počtu, co dovolovalo zařízení. Volba poloviny max. počtu je kvůli dalším HW omezením - při vyšších počtech vláken na blok nebylo dost registrů apod.

Druhá konfigurace měla každému dostupnému procesoru přiřadit právě jeden blok, počet vláken na blok se následně dopočítal. Tato konfigurace má nevýhodu v počtu vláken - může vznikout blok o malém počtu vláken, resp.

warp s několika málo vlákny. Další riziko bylo v přetečení počtu vláken v posledním bloku - na každé vlákno nemusela vyjít částice. Tento problém byl vyřešen přes podmínku a dopočet horní hranice, s důsledkem možného dělení warpu.

Cílem třetí konfigurace bylo rozdělit práci mezi bloky v násobcích maximálního počtu vláken v jednom warpu. Tj. počet vláken na blok byl daný, počet bloků se dopočítal. Pro třetí konfiguraci jsme zvolili 1 warp na 1 blok. Protože třetí konfigurace mohla mít vysokou režii kvůli počtu bloků, zkusili jsme čtvrtou konfiguraci s vyšším počtem warpů na blok - přesněji 4 warpy na 1 blok.

5.5 Výsledky měření

Výsledky měření jsou rozděleny dle konfigurací počtu bloků a vláken.

První konfigurace každému bloku přiřadila polovinu maximálního počtu vláken na blok. Počet bloků se dopočítal.

Měření	**Počet bloků**	**Počet vláken na blok**	**Čas**
1	2	512	40.412819
2	1	512	179.509688
3	8	512	19.369270
4	16	512	10.744051

Tabulka 22: Výsledky měření - první konfigurace

Druhá konfigurace vytvořila pro každý multiprocesor jeden blok, počet vláken na blok se podle toho dopočítával.

Měření	**Počet bloků**	**Počet vláken na blok**	**Čas**
1	15	68	35.196102
2	15	34	162.918234
3	15	273	13.931557
4	15	546	9.310848

Tabulka 23: Výsledky měření - druhá konfigurace

Třetí konfigurace vytvářela bloky podle maximálního počtu vláken ve warpu.

Měření	Počet bloků	Počet vláken na blok	Čas
1	32	32	34.932582
2	16	32	171.764859
3	128	32	16.723470
4	256	32	16.915611

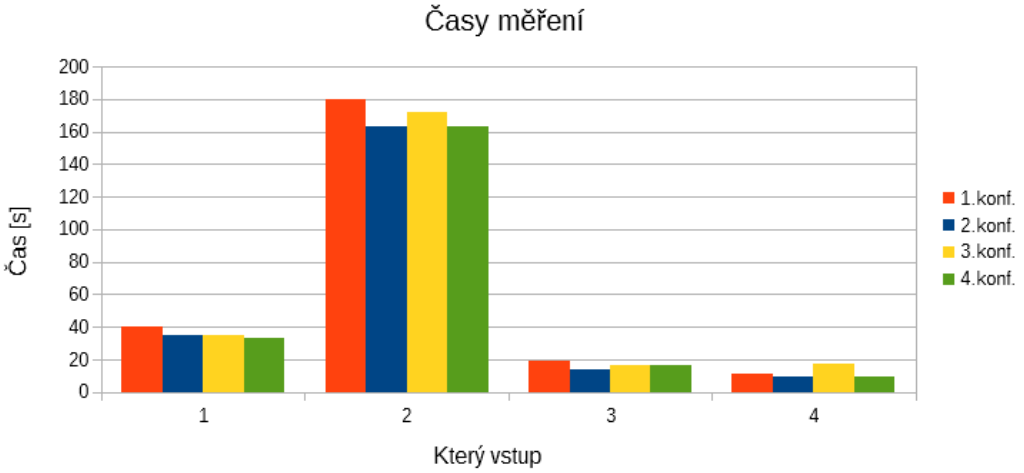
Tabulka 24: Výsledky měření - třetí konfigurace, 1 warp na blok

Třetí konfiguraci jsme upravili a každému bloku přiřadili čtyřnásobek maximálního počtu vláken v jednom warpu, výsledkem je čtvrtá konfigurace s následujícími výsledky.

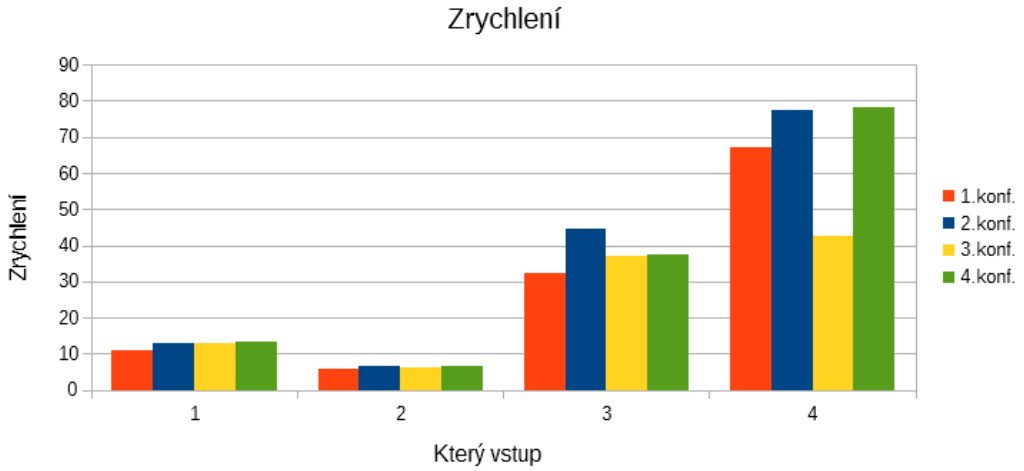
Měření	Počet bloků	Počet vláken na blok	Čas
1	8	128	33.198480
2	4	128	163.552016
3	32	128	16.599383
4	64	128	9.218899

Tabulka 25: Výsledky měření - čtvrtá konfigurace, 4 warpy na blok

Následují grafy časů měření a paralelního zrychlení. Paralelní zrychlení bylo počítáno vůči stejnému sekvenčnímu času, jako u výpočtu paralelního zrychlení pro Xeon Phi. Sekvenční čas je tedy z varianty 2 pro klasické procesory, tj. varianty s automatickou optimalizací a vektorizací.



Obrázek 13: Časy měření



Obrázek 14: Paralelní zrychlení

5.6 Vyhodnocení CUDA

Z výsledků měření je vidět, že algoritmu opět vyhovoval větší počet částic, v měřeních č. 3 a 4 bylo u všech konfigurací dosaženo nejvyššího zrychlení.

Co se konfigurací týče, nelze říci, že by jedna konfigurace byla optimální pro všechny vstupy. Nejlepší výsledky dosahují konfigurace č. 2 a 4, hodnoty paralelního zrychlení mají podobné.

Nejvyššího zrychlení dosáhla konfigurace č. 4, a to hodnoty až 78 v měření č. 4. Těsně za ní je konfigurace č. 2 se zrychlením 77 ve stejném měření. Konfigurace č. 4 dosáhla nejlepšího zrychlení ve 3 měřeních ze 4. Konfigurace č. 2 dosáhla nejlepšího zrychlení ve 3. měření (hodnota 44), v tomto měření měla konf. č. 4 zrychlení pouze 37. Lineárního zrychlení dosaženo nebylo.

Konfigurace č. 1, ve které se přiřazovala polovina maximálního počtu vláken každému bloku, měla problém s efektivním využitím všech dostupných multiprocesorů. To je vidět zejména na měření č. 1 a 2, ve kterých se využil 1, maximálně 2 multiprocesory. Naopak v měření č. 3 a 4 se muselo zapojit více multiprocesorů (zařízení jich mělo 15) a zrychlení již bylo výrazné, např. ve 4. měření dosahovalo hodnoty téměř 70.

Konfigurace č. 3, tedy 1 warp na 1 blok, trpěla kvůli vysokému počtu bloků s malým počtem vláken. Nejlépe je to vidět na 4. měření, kdy muselo být vytvořeno 8192/32 bloků, tedy 256 bloků pro 15 multiprocesorů. Při menším počtu částic se ještě dokázala vyrovnat nejrychlejším konfiguracím (měření 1, 2), ovšem již u 3. měření je vidět pokles a ve 4. měření je pouze lehce nadpoloviční hodnota zrychlení oproti nejrychlejší konfiguraci.

Výsledky konfigurace č. 2, tj. každému procesoru dát přesně 1 blok, byly víceméně očekávané - každý dostupný procesor byl využíván a práce byla rovnoměrně rozdělena. Výsledky konfigurace č. 4 jsou naopak poněkud překvapivé. Důvodem je pravděpodobně efektivnější provádění bloků, než tomu bylo ve variantě č. 2, kdy vznikaly warpy s menším než maximálním počtem vláken při každém výpočtu 1 kroku simulace.

6 Závěr

V rámci této semestrální práce byl implementován algoritmus NBody pro různé platformy, jmenovitě pro klasické Intel procesory, Xeon Phi a grafické karty nVidia. Nejprve byla vytvořena sekvenční implementace v jazyce C++, následovala optimalizace a vektorizace (automatická i ruční s využitím Intel AVX SSE2) a dále paralelizace použitím knihovny OpenMP. Poté jsme upravili program pro Xeon Phi a implementovali algoritmus za použití technologie CUDA.

Vizualizace simulace byla implementována pomocí knihovny CImg, nabízející živý výstup ze simulace přímo do okna programu. Další implementovaná možnost vizualizace je přes program gnuplot, kdy program generuje zdrojový kód a datový soubor pro gnuplot.

6.1 Porovnání výkonu všech implementací

Nejvyššího zrychlení dosáhla CUDA konfigurace č. 4 ve 4. měření, a to hodnoty 78. Xeon Phi dosáhl nejvyššího zrychlení 63 na stejné množině dat. V jednotkách času byl Xeon Phi v tomto měření pomalejší o přibližně 2 vteřiny. Paralelizace přes OpenMP na klasických procesorech dosáhla zrychlení až 14× s automatickou vektorizací, až 13,1× pak s ruční vektorizací. Pro připomenutí, automatická optimalizace a vektorizace zrychlila výpočet přibližně až 6×, po přidání ruční vektorizace až 8×.

Nutno podotknout, že hodnoty sekvenčního času, použité pro výpočet paralelního zrychlení pro Xeon Phi a CUDA, jsou brány z výsledků varianty 2 z implementace pro klasické procesory a tedy už jsou ovlivněny automatickou optimalizací a vektorizací, která výpočet zrychlila přibližně 6×. Pokud bychom počítali paralelní zrychlení za použití sekvenčního času z varianty 1 pro klasické procesory, výsledné zrychlení by bylo mnohonásobně vyšší.

6.2 Literatura

Při implementaci algoritmu jsme vycházeli zejména z článku uvedeném v zadání[2], pro implementaci algoritmu v jednotlivých technologiích jsme využili zejména přednášky předmětu MI-PAP, dále oficiální dokumentaci CUDA[3].

[2] `http://www.browndeertechnology.com/docs/BDT_OpenCL_Tutorial_NBody-rev3.html#algorithm`

[3] `http://docs.nvidia.com/cuda/`