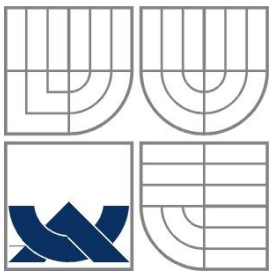
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

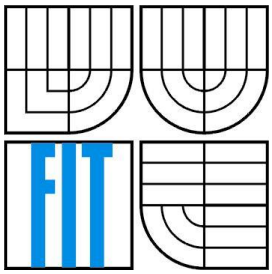
Fakulta informačních technologií
Faculty of Information Technology

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

Brno, 2016 Jiří Hanzlíček



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

SIMULACE VODY NA GPU

WATER SIMULATION USING GPU

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — Jiří Hanzlíček
AUTHOR

VEDOUCÍ PRÁCE — Ing. Filip Vaverka
SUPERVISOR

BRNO 2016

Abstrakt

Cílem této práce je najít vhodný model kapaliny, jehož numerickou simulaci lze realizovat jako interaktivní. Tento požadavek vede na řešení založené na vysoce paralelním algoritmu. Implementace je provedena na procesoru i na grafické kartě tak, aby bylo možné dosáhnout srovnání výpočetního výkonu jednotlivých zařízení na zvoleném modelu.

Abstract

The goal of this thesis is to find a suitable model of fluids, the numerical simulation of which can be realized as interactive program. This requirement leads to a solution based on highly parallel algorithm. The implementation is built not only for CPU, but also GPU in a way, that allows to compare computational performance of each device on selected model.

Klíčová slova

Interaktivní simulace, paralelizmus, OpenCL, GPU, Simulace kapaliny,
Smoothed Particle Hydrodynamics

Keywords

Interactive simulation, parallelism, OpenCL, GPU, Fluid simulation,
Smoothed Particle Hydrodynamics

Citace

Jiří Hanzlíček: Simulace vody na GPU, bakalářská práce, Brno, FIT VUT v Brně, 2016

Simulace vody na GPU

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Filipa Vaverky. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Jiří Hanzlíček
17.05.2016

Poděkování

Rád bych poděkoval vedoucímu bakalářské Ing. Filipovi Vaverkovi práce za pomoc při výběru tématu práce, odborné vedení a cenné rady při psaní bakalářské práce. Dále pak doc. Josefu Štětinovi za poskytnutí odborných studijních materiálů řešících problematiku hydrodynamiky. A v poslední řadě skupině neužších kamarádů, která mě podržela pracovní morálku v těch nejtemnějších chvílích při řešení této bakalářské práce.

Obsah

1 Úvod

Základy fyzikálního odvětví hydrodynamiky zabývající se mechanikou nestlačitelných tekutin byly položeny zformováním Navier-Stokesovy rovnice, kterou nezávisle na sobě zformulovali stejno jmenní vědci v roce 1845. Jedná se o rovnici, jejíž analytické řešení je dodnes známé pouze pro velmi omezený počet specifických případů. Byla dokonce zařazena mezi sedm takzvaných problémů tisíciletí, kde objevitel analytického řešení pro obecné použití bude štědře odměněn. V současné době je tedy nutné problematiku proudění kapalin nutné řešit numericky pro specifické případy. S nástupem číslicových počítačů se začaly formovat dva směry přístupu k modelování proudění. Jedním takovým to směrem jsou metody Eulerianovy a druhým metody Lagrangeho, kterými se podrobněji zabývá praktická část této bakalářské práce konkrétně modelem nestlačitelné kapaliny Smooth Particle Hydrodynamics.

Různé simulační modely je pak možné rozdělit podle případů jejich použití. Požadavky na jejich přesnost a míru abstrakce se pak budou odvíjet podle konkrétního použití. Například při studiích zabývajících se záplavami nebo protržení vodních nádrží bude kladen důraz na fyzikální přesnost modelu, tak aby bylo možné zmapovat potenciálně ohrožené oblasti. Nebo při simulačních testech různých strojírenských návrhů turbín a vysokotlakých zařízení. Touto oblastí simulací kapalin, kde je kladen důraz přesné hodnoty výpočtů se zabývají simulační prostředky spadající do kategorie CFD (Computational fluid dynamics). Jedním takovým program je například otevřený OpenFOAM. Oproti tomu při vizualizacích kapalin ve filmovém nebo herním průmyslu pro speciální efekty je nutné provádět vizualizaci v co nejkratších časech v případě herního průmyslu přímo v čase reálném.

Hlavní text práce je rozdělen do čtyř velkých kapitol, které jsou stejným dílem rozděleny jako kapitoly teoretické a praktické. První teoretickou kapitolou je kapitola 2 a zabývá se problematikou simulací na číslicových počítačích a základy hydrodynamiky, tak aby bylo možné pochopit mechaniku kapalin. V závěru kapitoly je pak popsána metoda simulování kapalin SPH. Druhá teoretická kapitola číslo 3 se zabývá popisem technologií vhodných pro implementaci simulace běžící v reálném čase.

Praktický blok začíná kapitolou 4, která popisuje požadavky kladené na implementaci a následně rozebírá její strukturu. Dále jsou v této kapitole obsaženy implementační zajímavosti. Závěrem praktické části je kapitola 5, ve které bylo provedeno měření výsledné implementace, jak rychle běží na různých zařízeních a za účelem zhodnotit do jaké míry výsledné řešení využívá potenciálu daného zařízení.

2 Modelování a simulace

Tato kapitola je zaměřena na stručný přehled pojmů modelování a simulace na číslicových počítačích. Definuje základní pojmy z problematiky a stanovuje základní klasifikaci modelů. Dále je nastíněna problematika simulace tekutin a popis jejich důležitých fyzikální vlastnosti z hlediska vizuálně věrohodné simulace. V poslední části kapitoly se nachází popis matematického principu modelu použitého pro vypracování praktické části této bakalářské práce.

2.1 Modelování

Modelování [1] je proces vytváření zjednodušeného modelu reálného systému na základě znalostí o něm. **Model** si pro účely simulace musí zachovat veškeré důležité části zobrazovaného systému, které jsou relevantní pro potřeby simulace, a ostatní nepotřebné vlastnosti vypouštět, nebo abstrahovat v co největší možné míře. Důležitost jednotlivých vlastností systému nelze jednoznačně stanovit, jejich významnost se odvíjí od účelu, za jakým je simulace prováděna. Takto vytvořený model reprezentuje formální znalosti o systému z hlediska ze zkoumaného hlediska. Lze tedy říci, že modelování je proces, při němž jsou transformovány znalosti strojově nezpracovatelné na reprezentaci proveditelnou na počítači.

Získání informací o modelovaném systému bývá spojeno s experimentováním s reálným vzorem systému. Znalosti poslouží nejen k vytvoření samotného modelu, ale i k jeho validaci. Validace modelu je proces, při kterém dochází k ověření modelu. Toto ověření se provádí tak, že model postavíme do známých situací reálného systému a pozorujeme, zda jeho reakce na podměty odpovídají reakcím systému reálnému (z předchozích experimentů).

Modely lze rozdělit do tří následujících kategorií [1]:

- **Modely spojité** se vyznačují spojitým časovým kontinuem, čili proměnné modelu mění svůj stav spojitě. Nejčastěji jsou tyto změny popsány pomocí diferenciálních rovnic. Všechny prvky takovéhoto systému pak reagují spojitě.
- **Modely diskrétní** se vyznačují tím, že se mění po skocích, kde jednotlivé skoky reprezentují reakce na konkrétní události.
- **Kombinované modely** kombinují prvky diskrétní tak spojité v jednom modelu.

2.2 Simulace

Simulaci [1] lze chápat jako vědeckou metodu, jejímž cílem je získání nových znalostí o systému experimentováním s jeho modelem. Pro získávání dat ze simulace musíme mít model validní, ne každý model je vhodný pro získání požadovaných dat.

Alternativou k simulaci je řešení modelu čistě z matematického hlediska analyticky, kde v ideálním případě jsou v podobě argumentů předány vstupní podmínky matematické funkci a jejím výsledkem je konkrétní hodnota. Bohužel tento způsob je vhodný pouze pro jednoduché systémy, kde je vytvoření matematického modelu pro systém v lidských silách. Pro vyřešení či sestavení komplikovanějšího popisu složitějších modelů však často není k dispozici dostatečný

matematický aparát. V souvislosti s prouděním tekutin se poté lze setkat se *Navierov-Stokesovu rovnicí*, jejíž analytické řešení pro obecný případ není známo. V závislosti na použití simulační metody a typu modelu lze klasifikovat simulace dle různých kritérií. Lze rozeznat tyto druhy simulací:

- Spojitá / diskrétní / kombinovaná simulace – je dělení simulací podle použitého modelu při experimentování.
- Kvalitativní / kvantitativní simulace – rozlišeno dle přesnosti získaných dat. Za kvalitativní simulaci je považována taková, u které není třeba větší počet opakování pro získání přesných dat z experimentu. Oproti tomu kvantitativní simulování průměruje a porovnává velké množství iterací simulace pro získání výsledku. Toho se využívá zejména u modelů, které nejsou přesně popsané.
- Simulace v reálném čase – jedná se o simulaci, která vyžaduje synchronizaci s reálným časem.
- Interaktivní simulace – vyžaduje aktivní zapojení experimentátora do procesu simulování. Umožňuje mu tímto dynamicky za běhu měnit vlastnosti simulace a analýzu stavu modelu.
- Paralelní distribuovaná simulace – obsahuje komplexní model, jehož simulace probíhá paralelně na více výpočetních zařízeních. Simulace proto musí kromě svého běhu zajistit také distribuci výpočetních úkonů a komunikaci mezi výpočetními jednotkami.

2.3 Modelování tekutin

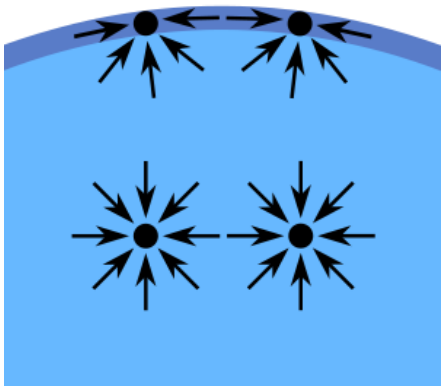
Tekutina [2] je jednoznačný název pro kapaliny a plyny, jejichž společnou vlastností je tekutost. **Tekutostí** je rozuměna schopnost látky se pohybovat jako celek i když nemá pevně vázané částice ve své struktuře. Takovýto pohyb se nazývá **tečení v proudu**. Tekutina je látka, která se na rozdíl od pevných látek nevratně deformuje a nemá svůj tvar. Tvar získávají působením vnějších sil a také podle svého okolí. Stěny ohraničující okraje tekutin se nazývají hladiny.

2.3.1 Fyzikální vlastnosti kapalin

Kapaliny jsou **nestlačitelné** tekutiny, tedy takové, které při působení vnějších sil mění svůj objem jen nepatrně, nebo vůbec. Pro potřeby modelování je podstatné popsat nejdůležitější vlastnosti kapalin, aby se dal formulovat model, který si klade za úkol co nejrealističtěji zobrazit virtuální kapalinu.

- **Hustota** kapalin se mění s okolním tlakem a teplotou jen nepatrně, a proto bývá při většině výpočtů považována za konstantní.
- **Viskozita tekutin** se projevuje při pohybu tekutin. Pohybují-li se sousední částice kapaliny rozdílnou rychlostí, tak zde dochází k takzvanému smykovému napětí, které pohyb brzdí. Částice pohybující se vyšší rychlostí, předávají svoji kinetickou energii částicím pomalejším a tím dochází k jejich brzdění. Naopak částice pomalejší jsou těmito silami urychlovány. Jedná se tedy o veličinu charakterizující vnitřní tření, které závisí především na vnitřních silách v kapalině a která napomáhá k ustálení stavu kapaliny. Čím vyšší je hodnota viskosity tím, je brzdění pohybu výraznější. Ideální kapalina by proto měla mít hodnotu nulovou.

- **Povrchové napětí** je způsobeno silami působícími mezi molekulami kapaliny. Uvnitř kapaliny je každá molekula obklopována stejnými molekulami a tím se jejich vzájemné působení vyrovnává. Oproti tomu na rozhraní jsou molekuly obklopeny jen ze strany jedné, a proto na hraniční molekuly působí výsledná síla směrem dovnitř kapaliny. Tento jev má za následek, že jednotlivé molekuly kapaliny se uskupují do větších celků[1] a na hladině vytvářejí nepatrnou bariéru s okolím.



Obrázek 2.1 Ilustrace působení sil mezi částicemi uvnitř a na okraji kapaliny. Převzato z [13]

2.3.2 Přístupy k vizuálně věrohodnému modelování kapalin v interaktivních systémech

Vytvořit vizuálně věrohodný model proudících kapalin je složitý proces, protože chování kapalin, tak jak je popsáno v předchozí sekci 2.3.1, vychází z jejich molekulární stavby. Povést modelovou abstrakci je poměrně složité až nemožné jako například u pohybu tuhých těles, kde možné těleso abstrahovat hmotným bodem. Nejvíce důvěryhodným způsobem pro zobrazení kapaliny by proto mohlo být modelování kapaliny jako systém modelů jednotlivých molekul. Jenže pouhý litr čisté vody H_2O obsahuje zaokrouhleně $3{,}3 * 10^{25}$ molekul. Avšak provést simulaci takového systému není na dnešním běžně dostupném hardwaru stále možné v uspokojivém čase. Navíc požadavky na simulace jsou tak náročné, že často vyžadují vizualizaci proudění několika kubíků vody, či dokonce proudění celých oceánů. Proto se každý model snaží dosáhnout co nejvyšší možné míry abstrakce pro zjednodušení výpočetní složitosti. To může přinést občas i nečekané problémy. Jedním z nejčastějších problémů při simulování kapalin, je problém udržet konstantní objem kapaliny. K tomuto problému dochází zejména při zaokrouhlování v průběhu aritmetických operací s čísly v plovoucí řadové čárce. Tato problematika je podrobněji diskutována v [3]. Ale také v důsledku aproximací samotné simulační metody při nepřesnostech, kdy byla nepřesně zvolena podoba numerické funkce (integrace / derivace).

V dnešní době se můžeme setkat s třemi hlavními směry abstrakce modelování tekutin:

- **Modely hladiny** zpravidla zanedbávají jevy probíhající pod hladinou kapaliny a kladou si za úkol co nejdůvěryhodněji popsat její chování, tedy šíření vln a ustálení hladiny. Jedním z nejznámějších

[1] Například jev zformování kapek deště.

modelů této kategorie je *Shallow Water* [4]. Největší výhodou těchto modelů je relativní výpočetní jednoduchost oproti následujícím bodům.

- **Modely založené na pevné mřížce** definují pevnou mřížku pro oblast simulace, ta musí být definována všude, kam se kapalina může potenciálně dostat.. Jednotlivé buňky simulace pak fungují jako spojené nádoby, mezi kterými dochází k přelévání kapaliny. Fungují tedy na principu celulárního automatu. Největší nevýhodou tohoto modelu je omezená velikost simulačního prostoru, jelikož jeho zvětšováním se zvyšuje i výpočetní složitost simulace. Základem této kategorie jsou *Eulerianovy rovnice* pro popis prodění tekutin v mřížce.
- **Modely částicové** zavádějí abstrahovanou částici tekutiny jako základní jednotku simulace. Takováto částice má podobné vlastnosti jako skutečné molekuly kapaliny s tím rozdílem, že jsou objemově větší, a tím se snižuje výpočetní složitost běhu simulace. Tyto částice se mohou volně pohybovat v prostoru a interagovat mezi sebou. Jednotlivé částice si nesou svoje data sebou a nejsou pevně uložena v mřížce. Na tomto základě může být tato kategorie řazena mezi *Lagrangianovy metody*. Hlavní výhodou těchto modelů je, že při práci s částicemi kapaliny nemůže dojít k jejich ztrátě a tudíž ke změně objemu kapaliny.

2.4 Model SPH

Smoothed-particle hydrodynamic (dále jen SPH) je metoda pro simulování proudění kapalin založená na částicích. Základní myšlenkou modelu je vyrovnávání relativní hustoty částic v prostoru. Každá částice má svoje skalární vlastnosti[2]. Vlastnosti kapaliny v určitém bodě tedy lze definovat pomocí částic v jeho okolí. Pro běh simulace se proto pro každou částici vypočítává relativní hustota na její pozici tak, aby bylo možné vypočítat místa s nižší hustotou a částice daným směrem posunout. K tomuto výpočtu se používá statistická metoda vyhlazovacích jader[3] k interpolaci hodnot vzdáleností. Použití vyhlazovacích jader napomáhá k plynulejšímu průběhu simulace bez extrémních výkyvů.

$$A(r_i) = \sum_{j}^{N} m_j \frac{A_j}{\rho_j} W\left(\left|r_i - r_j\right|, h\right) \tag{2.1}$$

- A relativní částicová hustota
- r částice v prostoru určena uspořádanou trojící souřadnic (x, y, z)
- N počet částic v systému
- m hmotnost částice
- ρ hustota kapaliny pojící se s částicí
- W funkce vyhlazovacího jádra
- h poloměr vzdálenosti ovlivnění

[2] Pozice v prostoru
[3] Z anglického názvu *Smoothing kernel*

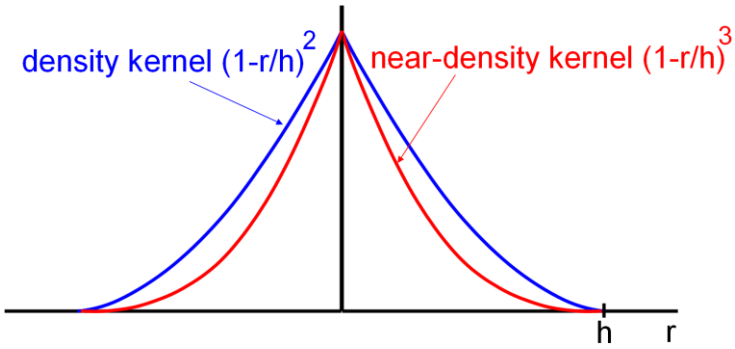
Předchozí rovnice (2.1) definuje podstatu principu modelu SPH. Určuje relativní hustotu částic A pro částici r_i na základě rozložení částic r[4] v systému. Výpočet dále může zohledňovat rozdílné váhy m_j pro jednotlivé částice. W je pak vyhlazovací jádro které interpoluje vzdálenost jednotlivých částic, ve vztahu k poloměru ovlivňované vzdálenosti h. Když je hodnota vzdálenosti dvou částic vyšší než hodnota h, tak jsou částice r_i a r_j vzdáleny natolik, aby se vzájemně neovlivňovaly. Pro potřeby této práce postačí zjednodušená verze rovnice (2.2), kde lze prohlásit, že všechny částice v systému mají stejné vlastnosti a tudíž hodnoty m_j a h za konstanty.

$$A(r_i) = \sum_j^N \frac{A_j}{\rho_j} W(|r_i - r_j|) \tag{2.2}$$

$$\nabla A(\vec{r_j}) = \rho_j \sum_j^N \left(\frac{A_i}{{\rho_i}^2} + \frac{A_j}{{\rho_j}^2} \right) \nabla W(|r_i - r_j|) \tag{2.3}$$

Z hodnot určujících relativní hustoty kapaliny na pozicích částic učených vztahem (2.2) je možné vytvořit gradient (2.3), který udává směr, kam částici přesunout tak, aby došlo k rovnoměrnému rozložení částic v modelu. Podobné rozptýlení částic je vidět na obrázku (Obrázek 2.1), který ilustruje jev vytváření hladiny (v kapitole věnující se povrchovému napětí). Tento základní SPH model navrhnuli B. Gindol a J. J. Monaghan v roce 1977 a zároveň nezávisle na nich i L. B. Lucy [5]. Model se pak dále vyvíjí a modifikuje do dnešní doby dle způsobů použití.

Model použitý v této práci se inspiruje variantou SPH modelu popsanou S. Clavetem a jeho kolegy v článku *Particle-based Viscoelastic Fluid Simulation* [6]. Jejich model rozšiřuje SPH o jev povrchového napětí tak, aby se částice kapaliny nejen rozprostíraly v prostoru, ale zároveň shlukovaly do větších celků. Autoři docílili tohoto jevu zavedením cílové relativní hustoty částic. Z toho vyplývá, že síly působící mezi částicemi nyní nejsou pouze odpudivé, ale i přitažlivé. To může způsobit problém s nekontrolovatelným shlukováním částic na jedno místo a jejich následným akcelerovaným rozprostřením, což vede k destabilizaci systému. K prevenci tohoto jevu autoři zavádějí druhou relativní hustotu částic pro blízké hodnoty, která při výpočtu výsledných sil nebere v potaz cílovou hustotu částic, a jejíž funkce pro výpočet relativní hustoty pro blízké částice má strmější průběh vyhlazovacího jádra. Jak je vidět na obrázku (Obrázek 2.2), účinek blízkých částic na výslednou váhu



Obrázek 2.2 Grafické znázornění průběhu vyhlazovacích jader
Převzato z [6]

[4] Částice r je definována pozici v prostoru

hustoty je stále nepatrně menší než složka vyvolávající shlukování, ale je dostatečně velká natolik tak, aby částice mezi sebou držely dostatečný odstup.

Výsledné tlaky působící mezi částicemi $p_{ij}^{vzdálený}$ a $p_{ij}^{blízký}$ jsou pak definovány rovnicemi (2.4) a (2.5), kde w je rychlost šíření vln v kapalině. Výsledné zrychlení $\vec{A}$ udělené částici na základě rozložení částic je pak definováno sumou (2.6).

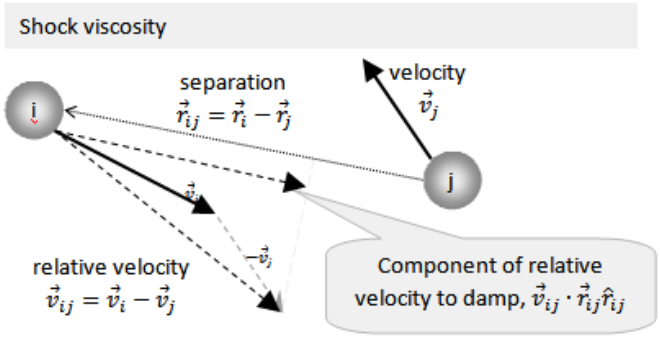
$$p_{ij}^{vzdálený} = w^{vzdálený\,2} \cdot \left(\rho_i^{vzdálený} + \rho_j^{vzdálený} - 2\rho^{cílová}\right) \cdot \left(1 - \frac{|r_i - r_j|}{h}\right)^2 \tag{2.4}$$

$$p_{ij}^{blízký} = w^{blízký2} \cdot \left(\rho_i^{blízký} + \rho_j^{blízký}\right) \cdot \left(1 - \frac{|r_i - r_j|}{h}\right)^3 \tag{2.5}$$

$$\vec{A}\,(r_i) = \sum_j^N \left(\left(p_{ij}^{vzdálený} + p_{ij}^{blízký}\right) \cdot \frac{r_i \cdot r_j}{|r_i - r_j|}\right) \tag{2.6}$$

- r částice v prostoru určena uspořádanou trojicí souřadnic (x, y, z)
- h poloměr vzdálenosti ovlivnění
- p tlak v kapalině
- ρ relativní částicová hustota
- $\vec{A}$ výsledná zrychlení udělené částici

Autoři [6] dále zavádějí do modelu vlastnosti vnitřního tření pro brzdění a ustálení pohybu částic. Tento jev je definován vlastností *radiální viskosita*, která je dána úhlem mezi vektory směru pohybu částic. Radiální viskozita tedy udává maximální velikost úhlu, kterou musejí svírat vektory rychlostí, aby se navzájem ovlivňovaly. Původní rychlost částice je následně ovlivněna poloviční hodnotou relativní rychlostí, jejíž výpočet je znázorněn na obrázku (Obrázek 2.3).



Obrázek 2.3 Ilustrace vektorového vyjádření protichůdných pohybů a výpočtu relativní rychlosti. Převzato z [14]

Tyto uvedené vlastnosti jsou velmi důležité pro popis chování kapalin tak, aby vypadaly důvěryhodně, jak je nastíněno v sekci 2.3.1. Z tohoto důvodu byl tento model zvolen pro implementaci praktické části této bakalářské práce.

3 Použité technologie

Tato kapitola se zabývá popisem jednotlivých nástrojů, které jsou použity na implementaci praktické části této bakalářské práce. Pro základní implementační prostředí byl zvolen jazyk *C++*, jednak z důvodu existence množství vhodných externích knihoven, jednak z důvodu různorodých možností optimalizace zdrojového kódu a plné kontroly nad prací s pamětí programu. Jazyk umožnuje nejen použití veškerých moderních implementačních přístupů, ale také povoluje nízko-úrovňové operace, které dnešní moderní jazyky často zastiňují před programátory. Díky těmto vlastnostem má programátor plnou moc nad svým kódem a jeho správným použitím může dosáhnout potřebné efektivity svých algoritmů.

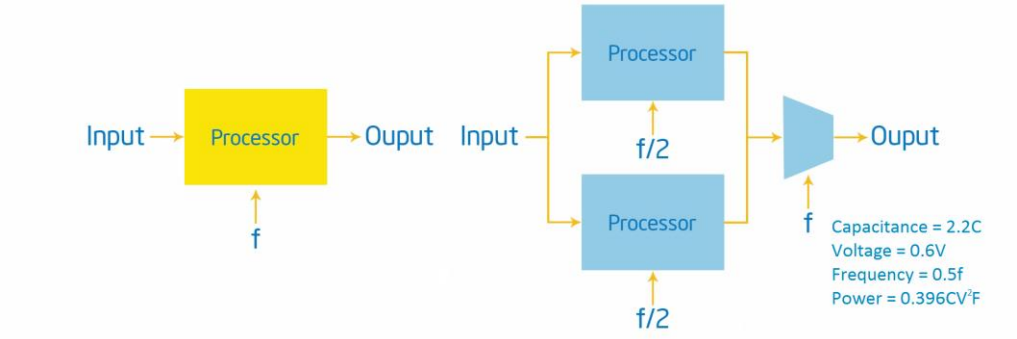
3.1 Paralelizmus

Výpočetní výkon počítačů se měří v jednotce *FLOPS* [5], která udává počet provedených operací v plovoucí řádkové čárce za sekundu. Z logického hlediska pak vyplývá, že zvyšováním kmitočtu procesoru lze docílit vyššího výkonu. Tento trend byl do nedávné doby jeden z hlavních směrů vývoje výpočetního výkonu. Z obecného výpočtu elektrického příkonu (3.1), kde C značí elektrickou kapacitu všech součástí (včetně vodičů ve výpočetní jednotce), U značí pracovní napětí a f kmitočet, vyplývá, že zvyšováním požadované frekvence musíme násobně zvyšovat příkon výpočetní jednotky. Se zvyšující se frekvencí, se pak častěji projevuje následek chvilkových zkratů, které vznikají při přepínání přepínaní tranzistorů v jednotlivých obvodech výpočetní jednotky. Větší četnost těchto zkratů pak vede k navýšení množství odpadového tepla. Bohužel dříve či později tato metoda dosáhne svého limitu. V současnosti vývojáři hardwaru narazili na problém, kdy při překročení limitu nejsou schopni efektivně zvýšit výpočetní výkon počítačů tímto způsobem.

$$P = CU^2 f \tag{3.1}$$

Předchozí odstavec nastiňuje základy problému, které jsou řešeny moderními trendy při paralelizací výpočtů pro dosažení vyššího výpočetního výkonu. Tato paralelizace zapříčiňuje umisťování většího množství jader do dnešních procesorů za cílem zvýšení výpočetního výkonu. Vyšší množství současně vypočtených hodnot znamená více vypočtených hodnot za jednotku času, stejně jako u navýšení frekvence výpočtu. A tento příztup ani nutně nemusí znamenat navýšení příkonu, jak dokazují autoři článku [7]. Princip, jak je tohoto možné dosáhnout, bude dále popsán a ilustrován pomocí zjednodušeného nákresu (Obrázek 3.1). Dosažení navýšení počtu jader navíc nemusí nutně znamenat technologický problém na základě *Moorova zákona*, který tvrdí, že co každých osmnáct měsíců budou vývojáři hardwaru schopni na procesoru zdvojnásobit počet tranzistorů.

[5] Zkratka z anglického Floating-point Operations per Second (počet operací v plovoucí řadové čárce za sekundu)



Obrázek 3.1 Ilustrace snížení příkonu pro stejný výpočetní výkon. Převzato z [15]

Na levé straně obrázku (Obrázek 3.1) je vyobrazen jednojádrový procesor pracující na frekvenci f. Ten tedy zvládá vypočíst N aritmetických operací (*FLOPS*). Oproti tomu na straně pravé je znázorněn procesor dvoujádrový, jehož jádra mají poloviční pracovní frekvenci. S nižší frekvencí se pojí možnost snížit pracovní napětí, jelikož není potřeba tak vysoký potenciál energie k rychlému přenosu náboje v jednotlivých komponentách procesoru. V pravém dolním rohu obrázku se nachází předpokládané vlastnosti procesoru dvoujádrové architektury tak, aby dosahoval stejného výkonu *FLOPS* jako procesor jednojádrový. Elektrická kapacita C celého procesoru je dvounásobně vyšší díky přítomnosti dvou jader a navíc je nutné zahrnout do celkového součtu propojující vodiče a řídící obvod. Proto je zde hodnota 2,2 násobku původní hodnoty kapacity. Frekvenci a napětí je možné snížit zhruba na polovinu, jak je popsáno výše. Výsledná hodnota příkonu paralelního procesoru je pak pod dosazení do rovnice (3.1) zaokrouhleně o šedesát procent nižší.

Výpočetní jednotka s vícejádrovou architekturou má následně teoreticky vyšší výpočetní výkon. Naneštěstí nelze algoritmicky jednoznačně stanovit pravidla, která by umožňovala jakýkoliv obecný zdrojový kód zpracovat paralelně. Chce-li programátor dosáhnout toho, aby jeho program efektivně využíval veškerý výpočetní potenciál, který mu hardware nabízí, musí k tomu přizpůsobit svůj zdrojový kód s vyžitím technologií k tomu určených[6]. V novodobé informatice, tak vznikl nový druh odvětví algoritmů pro paralelní programování. Lze také nalézt nemalé procento případů, kdy použití paralelního zpracování přináší paradoxně delší dobu zpracování oproti zpracování lineárnímu z důvodu velké režie řízení a synchronizace. O efektivitě vytížení hardwaru pak rozhodují především zkušenosti, znalosti a úsudek programátora.

3.2 GLFW

GLFW je otevřená multiplatformní knihovna pro vytváření oken s *OpenGL kontextem* a zachytávání vstupních událostí. Je napsána v jazyce *C* a umožnuje práci s okny v operačních systémech *Windows, OS X* a mnoho *Unixových distribucí* používající *Xserver*. GLFW dokáže reagovat na události okna pomocí takzvaných **call-back funkcí**. Ty může programátor definovat nejen na události vyvolané manipulací s oknem, ale i na události vyvolané periferními zařízeními, jako například stisknutí klávesy na klávesnici.

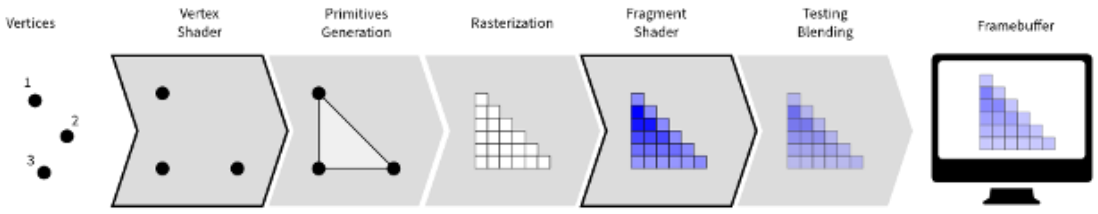
[6] OpenMP, OpenCL, nVidia Cuda, rozdělení aplikace do více procesů

3.3 OpenGL

OpenGL je otevřená grafická knihovina poskytující jednotné standardizované rozhraní pro práci s grafickým hardwarem. Standard rozhraní *OpenGL* je navržen tak, aby bylo nezávislé na hardwaru, a tím se dosahuje velké přenositelnosti kódu na rozmanité platformy. V současné době spravuje standard konsorcium *Khronos Group Inc.*

API OpenGL obsahuje pouze funkce na práci s grafickými primitivy v dvoudimenzionálním a třídimenzionálním prostoru a funkce na obsluhu hardwaru, nikoliv však na práci s okny. Mezi primitiva, která knihovna vykresluje, můžeme zařadit body, přímky a polygony. Z těch je možné následně sestavit libovolně složitý objekt k vykreslení. Veškerá data jsou ukládána v jednoduchých datových typech.

Samotné vykreslování je rozděleno do několika kroků, které popisuje standart OpenGL jako zobrazovací *pipeline* (v české literatuře pod názvem grafický řetězec). Veškeré výsledky vykreslování se ukládají do zobrazovací vyrovnávací paměti, kterou je nutné zobrazit po dokončení. Proces vykreslování lze ovlivnit na několika částech pipeline pomocí tzv. shader programů. Ty popisují operace provedené s daty v konkrétních částech vykreslování.



Obrázek 3.2 Ilustrace průběhu vykreslování v grafickém řetězci. Převzato z [16]

- **Vertex shader** zpracovává jednotlivé vrcholy určující primitiva (vertexy). Zde je možné provést nejrůznější prostorové transformace s vykreslovanými objekty bez narušení původních dat. Dokáže však pouze modifikovat jednotlivé body, tedy nevytvářet nové body, nebo je odebírat. Jedná se tedy o kód, který se vykonává paralelně pro každý vrchol nezávisle na ostatních. Nejčastěji se využívá k umístění jednotlivých objektů do souřadného systému scény vůči kameře.
- **Teselační shader** je nepovinnou součástí grafického řetězce. Slouží k rozdělení jednoho primitiva na několik menších.
- **Geometry shader** je další nepovinnou součástí grafického řetězce. Jedná se o poslední krok před rasterizací primitiv. Pouští se pro každé primitivum paralelně. Na rozdíl od Vertex shaderu má přístup ke všem vrcholům primitiva. Průběh Geometry shaderu může měnit typy a počet primitiv na výstupu oproti vstupu.
- **Fragment shader** přiřazuje barvu jednotlivým bodům objektu po rasterizaci. Výslednou barvu je zde možné skládat z několika textur a ovlivňovat ji pozicí vůči světlu. K výpočtu osvětlení se používají normály, které jsou modifikovány stejně jako pozice vrcholů ve Vertex shaderu. Rasterizované body je zde možné také úplně odebírat z výsledného zobrazení pomocí nastavení nulové hodnoty v alfa-složce barvy bodu.

- **Compute shader** je primárně určen pro paralelní výpočty velkého objemu dat. Díky své odlišnosti účelu použití je umístěn mimo grafický řetězec. Výsledek jeho výpočtů se neukládá do vykreslovací vyrovnávací paměti, jako je tomu při klasickém procesu vykreslení, ale do speciální vyrovnávací paměti shaderu. Data se nejčastěji předávají do výpočetního procesu v podobě textur. Z časového hlediska se jedná o předchůdce OpenCL API pro paralelní výpočty na grafickém hardwaru popsaném v následující kapitole. Jeho schopnosti jsou do značné míry omezeny z hlediska datové a řídící komunikace, ale zato má mnohem vyšší možnosti pro komunikaci s vykreslovacím řetězcem.

3.4 OpenCL

OpenCL je otevřený standard rozhraní pro paralelní programování na různých platformách, který je zaštítěn konsorciem Khronos Group Inc. Standard poskytuje jednotné programovací prostředí, které umožňuje psaní přenositelného kódu pro vícejádrové procesory, grafické karty a další hardware umožňující paralelní zpracování výpočtů.

OpenCL API je oficiálně vytvářeno v jazyce *C*, který je obalen vrstvou s *C++*. Komunitně dále vznikají adaptéry na další programovací jazyky jako například *Java*, nebo *Python*. Samotný jazyk OpenCL je syntakticky odvozen od jazyka *C* standardu *99* s rozšířením o vektorové datové typy a operace nad nimi. API dále umožňuje sdílení fyzických dat na zařízení s grafickým rozhraním *OpenGL*.

OpenCL je primárně navrženo pro řešení datově paralelních výpočtů. Tedy výpočtů, kde je nutné provést stejné aritmetické operace nad velkým objemem dat, a předpokládá se, že aritmetické operace budou převažovat nad prací z pamětí. Protože se stejná úloha vykonává souběžně na všech výpočetních jednotkách, není třeba provádět složité a časově náročné synchronizace, nestanoví-li programátor jinak. Specifikace standardu se rozdělují do čtyř následujících modelů:

- **Model platformy** – definuje vztah mezi hostitelským programem a zařízením OpenCL,
- **Exekuční model** – definuje abstraktní rozložení kernelů na fyzickém zařízení,
- **Paměťový model** – definuje způsob jak OpenCL pracuje s pamětí,
- **Programovací model** – definuje způsob mapování vláken na data.

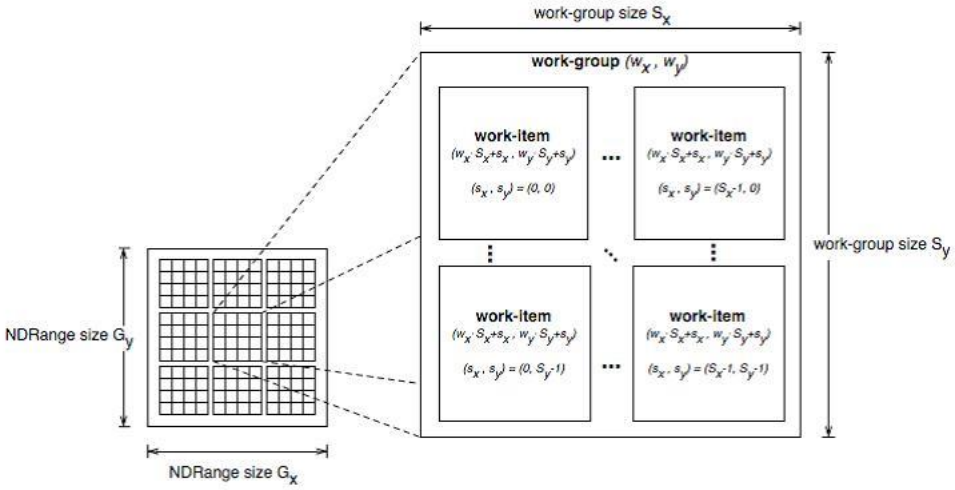
3.4.1 Model platformy

Stanovuje práci s API na nejvyšší úrovni. Zavádí pojem **hostitelského zařízení**, které řídí vykonávání programů na jednom nebo více zařízeních. Na hostitelském zařízení se se pracuje s **platformou** OpenCL, která reprezentuje komunikační rozhraní s hardwarem. Platformu dodávají samotní výrobci hardwaru. Z hostitele jsou odesílány řídící příkazy jednotlivým zařízením. Mezi nejčastěji používané příkazy patří nahrání dat z hostitele na výpočetní jednotku, spuštění série instrukcí (kernel) a stažení dat zpět na hostitelské zařízení.

3.4.2 Exekuční model

Určuje pojem *kernel*. **Kernel** je série instrukcí, která se má vykonat na výpočetním zařízení. Zdrojový kód kernelu na instrukce překládá hostitel. Spouštění kernelů se plánuje do příkazové fronty jednotlivých zařízení hostitelem.

Dále zavádí **N-dimenzionální**[7] abstraktní pracovní prostor výpočetních jednotek na zařízení. Takováto jednotka se anglické literatuře pojmenovává *work-item*. Jednotky se uskupují do pracovních skupin[8], podle vlastností konkrétního cílového zařízení. V rámci jedné pracovní skupiny se provádí stejný kernel paralelně na všech výpočetních jednotkách. Každá jednotka má přiřazený globální N-dimenzionální globální identifikátor, který jednoznačně určuje pozici jednotky v pracovním prostoru a N-dimenzionální lokální identifikátor, který jednoznačně určuje pozici jednotky v pracovní skupině. Podobným způsobem jsou přiřazovány identifikátory pracovním skupinám.



Obrázek 3.3 Ilustrace rozdělení 2-dimenzionálního pracovního prostoru. Převzato z [17]

3.4.3 Paměťový model

Zavádí paměťovou hierarchii, která popisuje různé paměťové oblasti na zařízení. Jednotlivé typy paměti jsou uvedeny v pořadí od nejpomalejší po nejrychlejší a zároveň nejmenší.

- **Globální paměť** – do této oblasti paměti mají přístup pro zápis a čtení všechny aktuálně běžící jednotky na zařízení. Hodnoty v paměti mohou být dočasně uloženy v rychlé vyrovnávací paměti. Při přístupu k této paměti je vhodné v rámci optimalizace přistupovat souměrně.
- **Konstantní paměť** – oblast paměti, která je přístupná všem běžícím jednotkám na zařízení k čtení, ale její obsah může měnit pouze hostitelské zařízení.
- **Lokální paměť** – oblasti paměti spojené s pracovní skupinou. Každá pracovní skupina má vlastní paměťový prostor, do kterého může ukládat a číst data.
- **Privátní paměť** – je paměť unikátní pro každou pracovní položku, která je odstíněná pro všechny ostatní. Je velikostně omezena počtem registrů výpočetní jednotky.

[7] Kde $N \in \{1,2,3\}$
[8] Ekvivalentní název z anglické literatury *work-group*

3.4.4 Programovací model

Zavádí dva modely, které definují, jak je možné zpracovávat data. Standard také nabízí možnost kombinovat tyto dva přístupy k modelům.

- **Datově paralelní model** definuje výpočet jako souběh instancí kernelů zpracovávajících data v rámci N-dimenzionálního prostoru zavedeného v Exekučním modelu. Na jednu instanci kernelu v tomto modelu připadá zpracovat jednu konkrétní datovou položku.
- **Úlohově paralelní model** umožňuje souběžné spuštění více rozdílných instancí kernelů, kde každý z kernelů je schopen zpracovat data velikosti jedné pracovní skupiny. Tento model je velmi podobný konvenčnímu paralelnímu zpracování na procesoru.

3.5 OpenMP

OpenMP je soustava direktiv pro překladač [8], které usnadňují vytváření vícevláknových programů v jazycích *Fortran, C* a *C++*. Standard OpenMP stanovuje způsoby, jak pracovat s pamětí a jakým způsobem je možné synchronizovat běh programu. Ideou programovacího modelu v OpenMP je existence hlavního vlákna pro řízení běhu programu. V částech kódu, kde je výhodné využít paralelní zpracování, je vlákno rozvětveno pro daný úsek. Úseky, kde dochází k větvení, jsou stanoveny programátorem direktivy.

Výhodou API je, že se skládá primárně z direktiv pro překladač tzv. *pragmat*. Překladače, které pragmata nepodporují, je tedy jen přeskočí a ponechají zdrojový kód lineární. Pro úsek ohraničený pragmatem lze stanovit v jeho argumentech vlastnosti vláken. Dále lze určit počet spuštěných vláken a nastavit sdílení paměti. Pro zajištění bezchybné synchronizace vláken se také zavádí *semafory* a popřípadě *bariéry* na sladění jejich běhu.

4 Návrh a realizace implementace

V této kapitole jsou popsány očekávané předpoklady, které by implementovaný model a samotná implantace měly splňovat. Dále je zde rozebrán model SPH z hlediska implementace modelu a popsány části výpočtů, jež mohou probíhat paralelně. Tato část dále pokračuje popisem, jak a po jakých operacích je nutné běh simulace synchronizovat. Pro testy na různých zařízeních musí být zdrojový kód přenositelný a mít možnost přeložení aplikace bez závislosti na grafickém API *OpenGL*. Jak je tohoto efektivně docíleno je popsáno v podkapitole Třídní rozložení implementace 4.2. Běh simulace velice často přepočítává velké množství vzdáleností, a proto je nutné zvolit správný přístup k tomuto výpočtu tak, aby zbytečně nebrzdil průběh celé simulace. Tomuto výpočtu je věnována vlastní sekce 4.3. Poslední část této kapitoly se pak zabývá problematikou generování náhodných čísel na grafické kartě v jazyce *OpenCL*, protože API neobsahuje vestavěné funkce pro generování pseudonáhodných čísel.

Cílem praktické části této bakalářské práce je naimplementovat matematicky náročný model popisující chování kapalin, pro účely demonstrace výhod paralelního zpracování. Zvolený model byl podrobně popsán v kapitole 2.4. Finální algoritmus pro výpočet má pak kvadratickou časovou složitost závisející na počtu částic v systému, jelikož pro výpočet relativní hustoty v bodě kapaliny je nutné provést sumu napříč všemi částicemi v systému. Množství aritmetických operací provedených v jednom simulačním kroku je tedy jednoduché ovlivnit počtem částic v systému, a tím se vliv přenáší i na aritmetickou složitost modelu. Model je implementován na CPU i GPU pomocí vhodných prostředků pro dosažení co nejefektivnějšího využití hardwaru, přičemž je zachována stejná posloupnost výpočtu tak, aby možné porovnat výpočetní výkon v tomto typu úloh obou typů zařízení.

Pro možnost ověření validního chování modelu musí být také implementován prostředek pro zobrazení modelu. Vizualizace simulace by měla být schopná zobrazit částice modelu a demonstrovat hladinu simulované kapaliny. Kvůli prezentaci funkčnosti modelu bude tedy implementováno interaktivní demo simulace, které bude v reálném čase reagovat na změny vyvolané uživatelem.

4.1 Rozčlenění SPH modelu na jednotlivé úkony

Model, který je podrobně popsaný v kapitole 2.4, byl zvolen jako vhodný kandidát pro vypracování paralelního algoritmu, jehož implementaci bude možné akcelerovat na různých výpočetních zřízeních. Následující seznam ve zkratce představuje chronologicky seřazené úkony, které je nutné vykonat pro každou částici v rámci jednoho kroku simulace:

- výpočet relativní částicové hustoty,
- výpočet gradientu tlaku částic,
- výpočet zrychlení udělený přetlakem částic v kapalině, popřípadě přitažlivými silami,
- zohlednění vnějších sil na částice,
- zohlednění vnitřního tření v kapalině,
- aplikace výsledných sil na rychlost částic,
- aktualizace poloh částic a případné kolize s okolím.

4.1.1 Použité vyhlazovací jádro

Volba vyhlazovacího jádra je jeden z důležitých kroků pro úspěšné vytvoření funkčního *SPH* modelu. Problematika této volby je podrobně popsána v článku [9]. Z něhož vyplývá, že funkce definující vyhlazovací jádro by mělo mít následující vlastnosti na intervalu $< -h\,;\ h >$:

- spojitá,
- sudá,
- hladká,
- omezená zdola, tak aby měla všechny hodnoty nezáporné,
- globální maximum na intervalu pro hodnotu 0.

Bylo experimentálně zjištěno, že jádro je zásadní části modelu. Aniž by došlo k narušení validního chování modelu[9], byla podoba vyhlazovacího jádra pro svoji důležitost a zásadní vliv na podobu chování kapaliny v modelu převzata od původních autorů použitého modelu [6]. Jejich použitý tvar je pak vyjádřen rovnicemi (4.1) a (4.2) jejíž grafické znázornění je vidět na obrázku (Obrázek 2.2).

$$W^{vzdálené}(r_i, r_j) = \left(1 - \frac{|r_i - r_j|}{h}\right)^2 \tag{4.1}$$

$$W^{blízké}(r_i, r_j) = \left(1 - \frac{|r_i - r_j|}{h}\right)^3 \tag{4.2}$$

- r částice v prostoru určena uspořádanou trojicí souřadnic (x, y, z)
- h poloměr vzdálenosti ovlivnění

4.1.2 Rozpis kroku simulace

Jednotlivé kroky výpočtu jednoho simulačního kroku jsou pojmenovány pomocí výčtového typu tak, aby názvosloví kódu bylo jednotné. Což umožňuje jednoduše provést záměnu pořadí, či přidání, nebo odebrání jednotlivých kroků. Pomocí tohoto výčtového typu pak dochází ke spouštění příslušných výpočetních funkcí na procesoru, nebo kernelů na grafické kartě. Jednotlivé části tohoto výčtového typu jsou níže popsány. Z funkčního hlediska je možné jej rozdělit do tří bloků oddělených prázdným řádkem. První takovýto blok obsahuje jednotlivé výpočetní úkony jednoho simulačního kroku, kde pro možnost pokračování ve výpočtu je nutné mít k dispozici výsledky předchozího kroku.

- `GENERATOR` je krok simulace, který vznikl z důvodu absence vestavěné funkce na generování pseudonáhodných čísel v jazyce *OpenCL*. Z tohoto důvodu je pro potřeby výpočtu v rámci každého simulačního kroku generován náhodný třírozměrný vektor pro každou částici simulace. Volba a způsob implementace generátoru jsou pak podrobněji popsány v samostatné podkapitole 4.4.

[9] Částice při experimentech s jinými vyhlazovacími jádry v průběhu simulace často vykazovaly validní chování, ale v některých těžko zrekonstruovatelných situacích došlo k neočekávanému shluknutí, či k rozprostření částic v simulačním prostoru.

- `CALCULATE_DENSITIES` je první krok samotného výpočtu, jehož účel vyplívá z jeho názvu. Jedná se tedy o určení relativní částicové hustoty. Je výhodné tento výpočet umístit do samotné funkce a jeho výsledek uložit do paměti, protože jeho výsledek je pak pro jednu částici N násobně použit v dalším kroku[10]. Matematické vyjádření tohoto kroku odpovídá rovnicím (4.1) a (4.2).
- `CALCULATE_PRESUREGRADIENT` po výpočtu relativních hustoty v kapalině je logicky možné pokračovat ve výpočtu gradientu tlaku. Tento výpočetní krok také zahrnuje výpočet výsledného zrychlení uděleného jednotlivým částicím v aktuálním simulačním kroku. Dochází zde tedy k výpočtu zrychlení částic na základě jejich vzájemné polohy v systému. Viz rovnice (2.4), (2.5) a (2.6). Stejně jako předchozí výpočetní krok, je i zde nutné provést výpočet pro každou částici v systému ve vztahu s každou ostatní.
- `APPLY_BODYFORCE` je první funkcí, která ovlivňuje samostatnou částici, nebere však v potaz interakci s ostatními částicemi. Účel této části simulačního kroku je aplikovat na pohyb částic vlivy prostředí. Primárně zde dochází k udělení zrychlení částicím z gravitačního pole, v kterém se nachází. Výsledné zrychlení je následně aplikováno na rychlost částice.
- `APPLY_SPEED_LIMIT` po modifikaci rychlosti jednotlivých částic je nutné pro zachování konzistence chování modelu zkontrolovat, zda některá z částic nepřekračuje fyzikálně možnou hranici. Pokud tomu tak je, je potřeba částice patřičně zpomalit.
- `VELOCITY_DIFFUSER` nyní když mají jednotlivé částice nové hodnoty rychlosti je vhodný okamžik pro aplikovaní pravidel vnitřního tření (popsaných podrobněji v kapitole 2.4). Pro detekci tohoto jevu, je nutné jej provést mezi všemi dvojicemi částic systému.
- `SPHERE_COLISIONS` pro zvýšení interaktivity a zajímavosti dema byl model chováni kapaliny rozšířen o kolize. Vhodné těleso pro nenáročnou implementaci je koule a to z několika následujících důvodů. Například množství dat uložené v paměti pro každou kouli v systému je poloha středu koule, jelikož poloměr mají všechny koule unifikovaný. Hlavní výhodou je pak jednoduchost samotné detekce kolize, jelikož pro její zjištění stačí pouze kontrolovat pozici částice a pozici středu vůči poloměru koule a částice. Problematikou střetu vektoru a koule se pak věnuje rozmanité množství zdrojů v oboru fotorealistického zobrazení. Proto bylo původní ideou pro princip výpočtu průsečíku a odrazu, použít existující jednoznačné a optimalizované řešení. Aplikace existujících řešení v praxi pak ovšem ukázala, že v tomto případě je matematická přesnost na škodu. Částice v okolí koulí se pak dostávaly do oscilujících a nevěrohodně vypadajících stavů. Výsledná implementace má tedy k matematickému vyjádření odrazu dvou koulí poměrně daleko, ale splňuje veškeré potřebné předpoklady a přináší pro model mnohé výhody. Hlavní výhodou zvoleného postupu je zjednodušení aritmetické složitosti výpočtu a tím i snížená náročnost na počet registrů ve výpočetní jednotce. Další výhodou je již zmíněné řešení problému s oscilací částic. Výpočet je pak dále rozšířen o podmínku, která zajišťuje, aby každá částice efektivně sklouzla pomocí pseudonáhodných čísel z povrchu koule a nezůstávala na ní. Částice se tedy ve výsledné implementaci od povrchu koule odrážejí a nejsou schopny koulí propadnout.

[10] Kde N rovná se počtu částic v simulaci

- `APPLY_VELOCITY` po zohlednění všech vlivů na rychlost nezbývá nic jiného než samotný pohyb aplikovat na samotné částice. Aby nedošlo k úplnému rozlití kapaliny po nekonečném prostoru, byla vytvořena neviditelná hranice ohraničující simulační prostor. Při kolizi s bariérou dochází k odrazu a částečnému pohlcení rychlosti.
- `CUBES_OCCUPATIONS` funkce sloužící k vizualizaci hladiny kapaliny. Pro tyto účely, byl pracovní prostor rozčleněn do kubické mřížky. Jednotlivé buňky jsou zobrazovány pod podmínkou, že se v nich nachází požadované množství částic kapaliny. Nevýhodou tohoto řešení je, že omezuje hlavní výhodu částicových modelů, a to nezávislost na velikosti simulačního prostoru. Možnosti jiného zobrazení jsou pak diskutovány v kapitole 6.

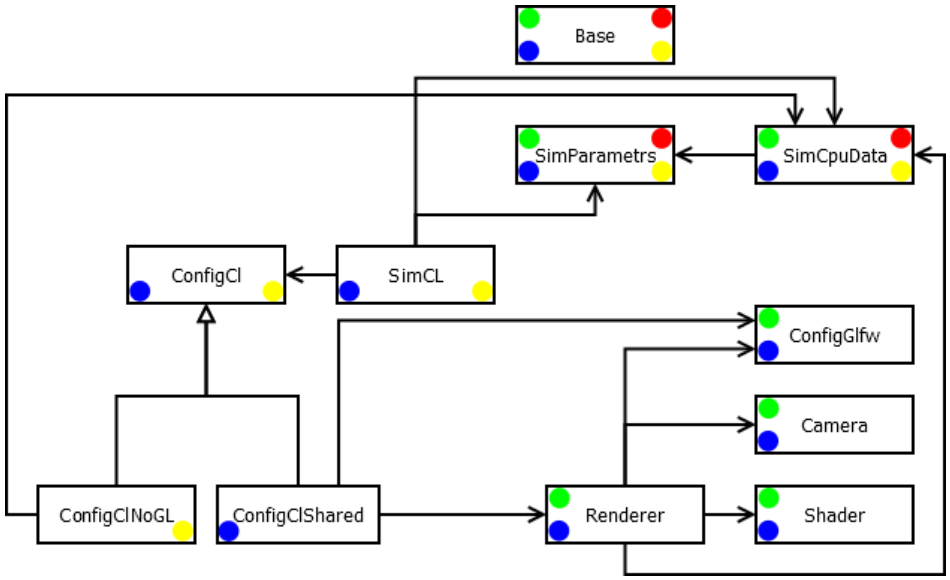
Druhý blok pak obsahuje názvy zbylých *Kernel programů*, které nejsou periodicky pouštěny v průběhu simulačních kroků. Jejich přítomnost v tomto výčtovém typu není tedy z logického hlediska nutná, ale pro zachování jednotné podoby obsluhy jednotlivých *Kernelů* zde byly ponechány.

Poslední blok pak vyzdvihuje důležitost a použitelnost zavedení tohoto výčtového typu. Jednotlivé položky posledního bloku fungují jako zarážky pro řídící cyklus simulace. Tato myšlenka původně vznikla jako jedna z možných implementací cyklu *foreach*. Výhoda zvolené varianty se projevila pak v okamžiku potřeby vyloučení položky `GENERATOR` ze simulačního cyklu implementovaného na CPU. Mimo jiné velice snadno umožňuje vložení nových kroků simulace mezi jednotlivé části výpočtu, nebo změnu jejich pořadí.

4.2 Třídní rozložení implementace

Pro potřeby možnosti spustit simulaci na různých zařízeních, tak aby mohlo dojít k jejich porovnání, bylo nutné vytvořit přeložitelné kódy nezávislé na knihovnách *OpenGL* a *OpenCL*. K tomu, aby nebylo nutné přepisovat několikrát celý zdrojový kód pro jednotlivé případy užití, byla implementace rozdělena do několika tříd. Z nich je pak možné sestavit čtyři rozdílné programy:

- `ibp` je primárním výstupem praktické části této bakalářské práce a reprezentuje interaktivní demo akcelerované pomocí grafické karty, jež vykresluje výstup simulace. Jedná se tedy o program závisející na obou výše zmíněných knihovnách.
- `openClNoGL` je verze programu, která nemá grafický výstup. Vznikla primárně pro potřeby měření výpočetního výkonu jednotlivých zařízení, tak aby nedocházelo k zatěžování hardwaru vykreslováním výstupu simulace.
- `singleCpu` je interaktivní demo simulace, jehož výpočet probíhá na procesoru.
- `singleCpuNoGL` vznikla podobně, jako verze `openClNoGL` pro potřeby měření výkonosti procesoru za použití různého počtu jader.



Obrázek 4.1 UML tříd implementovaných v rámci praktické části.
modrá – ibp
žlutá – openClNoGL
zelená – singleCpu
červená – singleCpuNoGL
V rámci zvýšení přehlednosti byly odstraněny šipky závislostí na base souboru

V rámci implementace vzniklo 11 hlavičkových souborů, 14 *Cpp* souborů, 7 shader souborů a jeden cl soubor. Obrázek 6.2 zobrazuje závislosti mezi jednotlivými třídami implantace. Barevná kolečka u jednotlivých položek znázorňují pro překlad jaké verze implementace je daná třída potřeba, tak aby nevznikaly nadbytečné závislosti. Níže následuje popis těch pro simulaci nejdůležitější a implementačně zajímavých:

- Veškeré soubory v implantaci jsou závislé na předkompilovaném hlavičkovém souboru `Base`. Ten importuje nejnutnější knihovny a definuje velikost pracovního simulačního prostoru. Dále obsahuje výčtový datový typ s chybovými kódy, které mohou nastat v průběhu inicializace a běhu simulace.
- `ConfigCl` je bázovou třídou pro inicializaci *OpenCL* prostředí na hostitelském zařízení. Obsahuje důležité prvky pro řízení *OpenCL* programů na zařízení. Vytváření kontextu však musí být metodou virtuální, jelikož v okamžiku použití vykreslovací knihovny *OpenGL* je nutné provést inicializaci na známém zařízení, tak aby mohlo dojít k propojení paměti mezi grafickým řetězcem a výpočetním kontextem. Toto zajišťuje třída odvozená `ConfigClShared`. Oproti tomu v okamžiku kdy nedochází k použití *OpenGL* paměti, je nutné vytvořit paměťové prvky vlastní, stejně jako provést volbu výpočetního zařízení. Pro tyto potřeby byla vytvořena další odvozená třida `ConfigClNoGL`.
- `SimParameters` je třída uchovávající informace o simulované kapalině a simulačním prostředí. Od parametrů uložených v této třídě se odvíjí celkové chování modelu kapaliny.

`m_nearToFar`	Modifikátor, který určuje rozdíl mezi vzdálenostmi poloměru ovlivňování pro částice blízké a vzdálené.
`m_stiffness`	Ovlivňuje tekutost kapaliny a tím rychlosti číření vln v kapalině a maximální možnou dosažitelnou rychlost pro jednotlivé částice.
`m_particlesRadius`	Určuje velikost částic v simulaci.
`m_influenceConst`	Určuje poloměr vzdálenosti ovlivňování.
`m_fluidDensity`	Fyzikální vlastnost definující chování kapaliny.
`m_targetNumberDensity`	Hodnota relativní částicové hustoty, které se snaží částice v kapalině dosáhnout, tak aby došlo k rovnoměrnému rozložení částic. Tato vlastnost určuje velikost sil způsobující shlukování.
`m_radialViscosity`	Hodnota prostorového úhlu, který určuje, do jaké míry se bude projevovat vnitřní tření částic v kapalině. Čím je jeho hodnota větší, tím více částic se bude navzájem ovlivňovat.
`m_gravity`	Určuje velikost gravitačního pole, ve kterém se simulační prostředí nachází.
`m_ambientDensity`	Určuje Fyzikální hustotu simulačního prostředí, společně s hustotou kapaliny určují cílovou relativní hustotu částic.

Tabulka 4.1 Rozpisjednotlivých vlastností s jejich významem na simulaci

- `SimCpuData` je třídou obsluhující data a výpočty na procesoru. Data uložená v této třídě slouží jako zdrojová pro kopírování dat při inicializaci paměťových prvků *OpenCL* a *OpenGL*.
- `SimCl` je třída obsluhující běh simulace na jednotlivých zařízeních. Obsahuje inicializace jednotlivých paměťových prvků a kernelů. Ty je pak schopná pouštět a měřit dobu jejich provedení.
- `Renderer` je třída zajišťující vykreslování simulace. Jednotlivé částice simulované kapaliny jsou vykreslovány pomocí bodů, které mění svoji barvu v závislosti na tlaku v kapalině, čím je tlak v pozici bodu vyšší, tím se brva více přibližuje k červené barvě. Informace o tlaku kapaliny je vypočítána v simulačním kroku `CALCULATE_PRESUREGRADIENT` odkud je pomocí sdíleného *glBuffuru* předán fragment shaderu pro vykreslování částic. `Renderer` dále vykresluje jednotlivé koule a kostky reprezentující hladinu kapaliny. K vytvoření dobře vypadajících koulí je pak využit *geometry shader*, který rozděluje jednotlivé vertexy kostky a pomocí goniometrických funkcí je dělí na menší díly.

4.3 Optimalizace výpočtu vzdálenosti

Primární části výpočtů v modelu SPH vychází ze vzdáleností jednotlivých částic. Výpočet pro vzdálenost dvou bodů $\vec{a}$ a $\vec{b}$ v N-dimenzionálním prostoru je definován funkcí (4.3). Tato funkce však obsahuje odmocninu, což je pro výpočet v plovoucí řádové čárce jedna z nejdéle trvajících operací. Jazyk *OpenCL* disponuje dvěma vestavěnými funkcemi pro výpočet vzdálenosti a to `distance` a `fast_distance`. Jedna metoda je přesnější a druhá rychlejší, jak naznačuje její název.

$$vzdálenost(\vec{a}, \vec{b}) = \sqrt{\sum_{i=1}^{N} (a_i - b_i)^2} \tag{4.3}$$

Na volbu kandidáta pro použití v implementaci modelu byl vytvořen jednoduchý úkol, který slouží k výběru nejefektivnějšího přístupu výpočtu. Úkolem testovacích funkcí je sečíst vzdálenosti mezi všemi částicemi, které jsou v poloměru dosahu vyhlazovacích jader. Tím dojde k využití výsledných vzdáleností a optimalizace překladače tedy nemohou výpočet odstranit z binárního kódu. Čas k výchozímu srovnání stanovuje funkce, která počítá vzdálenosti pomocí matematických funkcí. V implementaci ji pak lze nalézt pod názvem `DistTest1`. Tato funkce nejprve vypočte vzdálenost mezi dvojicí částic, a pokud je vzdálenost mezi nimi menší než poloměr dosahu vyhlazovacích jader, je hodnota vzdálenosti přičtena k celkovému součtu. Na stejném principu pak pracuje funkce `DistTest2`, která používá k výpočtu vestavěnou funkci `distance`. Stejně tak `DistTest3` využívá funkci `fast_distance`. V rámci poslední testovací funkce `DistTest4` je výpočet odmocniny přesunut až do těla podmínky před samotné přičtení. Takto může být učiněno na základě matematické ekvivalence porovnání druhých mocnin vzdáleností. To znamená, že funkce pro porovnání poloměru působnosti využívá jeho druhou mocninu vůči části výpočtu vzdálenosti nacházející se pod odmocninou. Odmocnina se tedy počítá pouze až v případě, kdy je opravdu nutné znát vzdálenost mezi částicemi.

Počet částic	100	1024	10240	20480
`DistTest1`	0.04099 ms	0.34541 ms	7.42493 ms	29.89482 ms
`DistTest2`	0.05971 ms	0.53904 ms	15.86554 ms	63.30787 ms
`DistTest3`	0.04090 ms	0.34448 ms	7.42506 ms	29.90531 ms
`DistTest4`	0.03725 ms	0.31104 ms	6.47763 ms	26.45408 ms

Tabulka 4.2 Měření doby trvání výpočtu vzdálenosti pro různý počet částic na stejném pracovním prostoru. Měření proběhlo na grafické kartě NVIDIA GeForce GTX 960M

Z tabulky (Tabulka 4.2) vyplývá, že nejvýhodnějším způsobem výpočtu vzdálenosti mezi částicemi je varianta `DistTest4`, tedy dopočítávat přesnou hodnotu vzdálenosti až v okamžiku, kdy je skutečně potřeba pro potřeby výpočtu. Zajímavostí v tabulce je skutečnost, že vestavěná funkce `distance` pro zvýšení přesnosti výpočtu spotřebuje velké množství času.

4.4 Generování náhodných čísel na GPU

Jak již bylo dříve zmíněno, jazyk OpenCL neobsahuje vestavěné funkce na generování pseudonáhodných čísel. Model SPH ovšem pro svůj běh náhodná čísla potřebuje v okamžiku, kdy jsou částice v kolizní vzdálenosti, aby nedošlo k výpočtu enormních hodnot tlaku. Další výhodou tohoto opatření je rozbití matematické přesnosti modelu, jelikož v určitých případech mohlo dojít k uvěznění částic do mřížky, ve které se síly mezi částicemi vyrušily a tvořily tak nevalidní fragmenty kapaliny. Tento problém byl v rámci implementace vyřešen vytvořením kernelu, který naplní pro každou částici třírozměrný vektor pseudonáhodnými hodnotami. Ten pak slouží k posunutí částic v systému.

Pro jazyk OpenCL existuje poměrně malý počet rozšíření v podobě různých generátorů od vývojářů třetích stran. Tyto rozšíření se pak vyznačují použitím bezpečnostně kvalitních a nepředvídatelných algoritmů[11], ty však velmi často vyžadují nemalé množství uložených dat v paměti, které v průběhu modifikují a z nich skládají výsledné číslo. Při náhodném posunu jakékoli částice v modelu by tak bylo potřeba uložit instanci generátoru pro každou částici v paměti. Jiné typy generátorů pak využívají komunikace s procesorem, což v případě velkého množství částic značně prodlužuje režii obsluhy. Z těchto důvodů bylo rozhodnuto implementovat generátor vlastní.

K použití pseudonáhodných čísel dochází v nepravidelných intervalech pro různé částice. Tento vztah závisí čistě na poloze částic v systému. Z tohoto důvodu bylo pro zachování jednotné podoby běhu simulace rozhodnuto, že v každém kroku se vygeneruje sada čísel pro každou částici tak, aby nedošlo k nadstandardní době trvání spuštěných kernelů, a to i za cenu, že nebudou použity. Náhodné hodnoty je tedy nutné generovat rychle, s malou paměťovou zátěží a s alespoň minimální nepravidelností následujících hodnot. Tyto podmínky skvěle splňuje jeden z prvních a nejznámějších generátorů *lineární kongruentní generátor*.

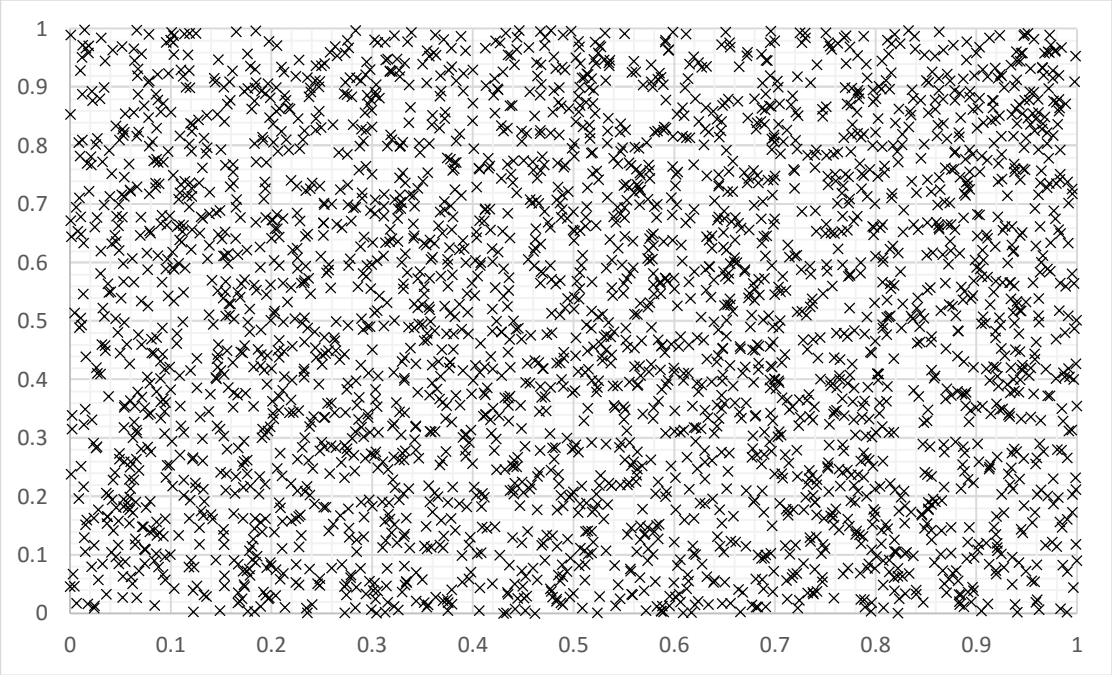
$$X_{n+1} = (aX_n + c) \bmod m \tag{4.4}$$

Princip lineárního kongruentního generátoru je vyjádřen v rovnicí (4.4). Pro výpočet náhodného čísla potřebuje výsledek předchozího výpočtu nebo inicializační hodnotu[12] a vhodné konstanty definující povahu generátoru. Hodnota nového X náleží do intervalu omezeným konstantou m. K implementaci operace modulo se nejčastěji využívá možnost přetečení neznaménkového datového typu. Typickým kandidátem bývá neznaménkový třiceti dvou bitový celočíselný datový typ[13]. Této vlastnosti bylo využito i při implementaci vlastního generátoru. Volba konstant a a c byla provedena podle [10] tak, aby nedošlo ke generování čísel v pravidelném a omezeném rozsahu. Použité konstanty pak mají následující hodnoty:

- $a = 1103515245$
- $c = 12345$
- $m = 2^{32}$

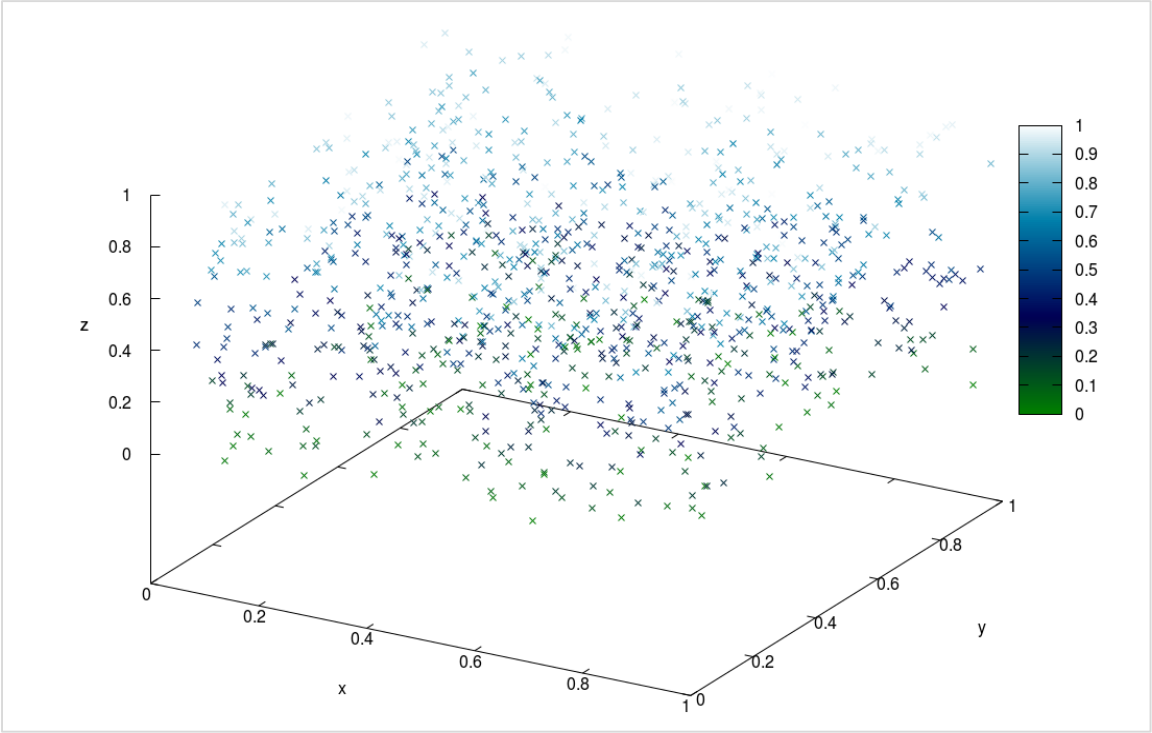
[11] Nejčastěji některá z modifikací generátorů *Mersenne Twister* nebo *Well*
[12] Ekvivalent často používaného výrazu *seed*
[13] V jazyce C++ `uint32_t` z knihovny `cstdint`

Častým problémem špatně fungujícího pseudonáhodného generátoru je nerovnoměrné rozložení generovaných hodnot či nízká perioda. Pro ověření jejich správné funkčnosti se pak nejčastěji používá vizuální kontrola výstupních dat pomocí grafů znázorňujících vygenerované uspořádané n-tice v podobě bodů. Alternativně lze také náhodná čísla testovat statistickými metodami, kde vygenerovaná čísla jsou podrobena matematické analýze. Například série testů *diehard* nepodrobuje kontrole pouze vygenerovaná čísla, ale přistupuje k nim jako k binárním datům. Data pak ukládá do různých datových struktur a hledá souvislosti mezi nimi a jejich částmi. Pro test zvoleného lineárního kongruentního generátoru proto byly vytvořeny následující grafy Graf 4.1 a Graf 4.2.



Graf 4.1 Grafické znázornění výstupu zvoleného generátoru 2D.

Graf 4.1 představuje vizuální výstup generování tři tisíc dvojic, které vytvořil implementovaný generátor, jehož vygenerované hodnoty byly převedeny na interval $< 0 ; 1 >$. Jak je na grafu patrné, jsou body v prostoru rovnoměrně rozloženy bez náznaku vytváření pravidelných obrazců. U některých generátorů pak dochází k objevení chyby přidáním dalších dimenzí. Z důvodu, že pro běh simulace je třeba generovat třírozměrný vektor, je nutné ověřit jeho chování i v třetím prostoru.



Graf 4.2 Grafické znázornění výstupu zvoleného generátoru 3D

Graf 4.2 zobrazuje tisíc vygenerovaných trojic[14]. Vizuální kontrolou je patrné, že zde také nedochází zásadním nedostatkům. Testy náhodného generátoru byly provedeny na různých intervalech s rozdílnými počty částic. Zobrazené grafy pouze demonstrují princip ověřovacího procesu a byly zvoleny jako názorné ukázky funkčnosti implementovaného generátoru pseudonáhodných čísel.

[14] Počet bodů byl omezen pro zvýšení přehlednosti

5 Měření vytížení hardwaru

V rámci řešení této práce není provedená pouze implementace datově paralelního výpočtu simulace kapaliny, ale také bylo provedeno měření výsledné implementace, jak rychle běží na různých zařízeních a za účelem zhodnotit do jaké míry výsledné řešení využívá potenciálu daného zařízení. V První části této kapitoly je popsáno jak měření a výpočty probíhaly. V následující kapitole je pak znázorněn princip přidávání výpočetních jader na procesoru a vliv tohoto jevu na průběh simulace. Následuje statistika využití paměťových propustností a využití výpočetního potenciálu na jednotlivých zřízeních.

5.1 Popis měření

V rámci měření implementované aplikace byly měřeny tři směrodatné hodnoty, doba provedení jednotlivých výpočetních úkonů v rámci jednoho simulačního kroku, teoretické využití propustnosti paměti a teoretické využití výpočetního výkonu. Veškerá měření probíhala v sériích o různém počtu částic na všech zařízeních. Počet částic při každém testu pak byl vždy mocninou čísla dvě tak, aby mohlo dojít k zarovnanému mapování vláken či kernelů na cílový hardware.

Měření doby výpočtu na procesoru probíhalo pomocí systémové knihovny `chrono`. Oproti tomu při použití platformy *OpenCL* je nutné pro získání doby provedení kernelu nástroj k tomu určený *ClEvent*. Princip měření je následně pro obě technologie stejný, poznačit si čas na začátku operace a na jejím konci a rozdíl těchto dvou časů dává dobu jednoho kroku. Takto získaná data jsou v průběhu simulace sčítána a na jejím konci zprůměrována tak, aby byl získán průměrný čas provedení jednoho kroku výpočtu simulace. Pro získání validních hodnot je nutné zvolit vhodný počet simulačních kroků. Počet kroků nesmí být příliš malý na to, aby nemohlo dojít k nějakým zásadním výkyvům v důsledku inicializačního náhodného rozložení částic v simulaci. Experimentálně byl počet korků pro potřeby měření stanoven na 250. Výstup jednoho měření lze vidět v příloze C. Data takto získaná byla dále zpracovávána v programu Microsoft Office Excel 2016.

	čtení		**zápis**	
	float / int32	float3	float / int32	float3
CalculateDensities	3*N	0	2	0
CalculatePressureGradient	5*N	1	1	1
ApplyBodyforce	0	2	0	1
ApplySpeedLimit	0	1	0	1
VelocityDiffuser	3*N	N	0	1
SphereColisions	3+3*K	2	3	1
ApplyVelocity	3	1	3	1
LCG	1	0	1	1

Tabulka 5.1 Výpočet propustnosti paměti
Kde N udává počet částic v systému a K počet koulí

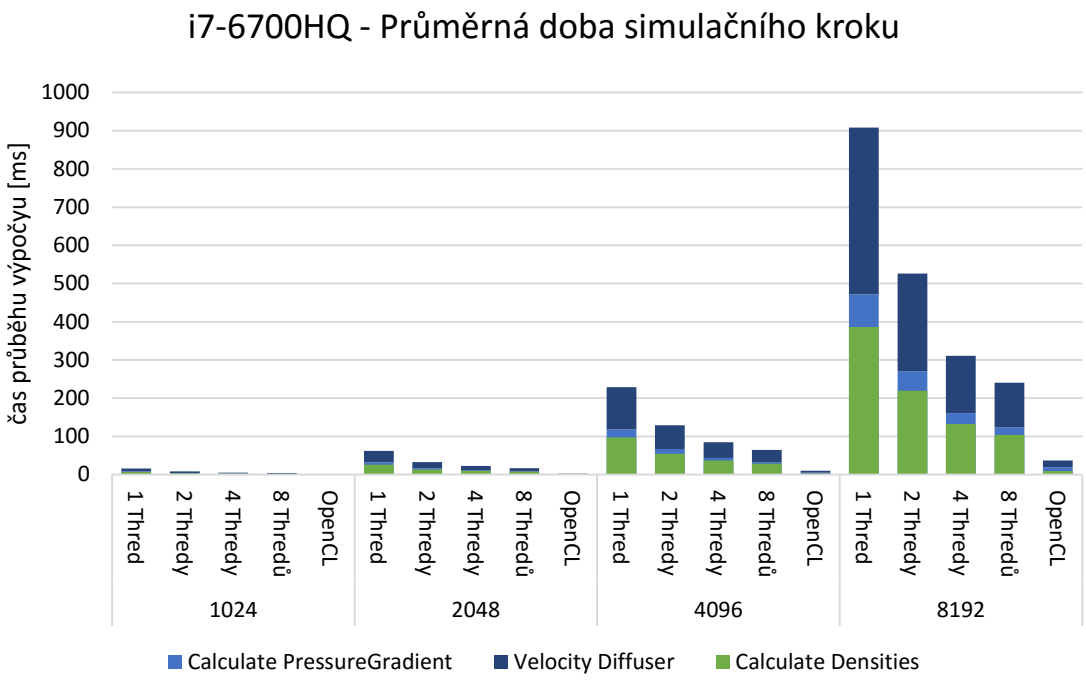
Tabulka 5.1 znázorňuje počet paměťových operací, které se musejí provést při jednom simulačním kroku pro jednu částici. Jedná se tedy o objem dat, které je nutné přenést pro jednu částici simulace. V případě *OpenCL* implementace tyto čísla odpovídají jednomu kernelu. Počty uvedené v této tabulce vycházení přímo ze zdrojového kódu jazyka *OpenCL* v souboru `sph.cl`. Propustnost paměťových prvků se udává v jednotce *GBPS (Giga Byte Per Second*[15]*)*. Tedy v počtu přenesených dat za jednotku času. Výpočet využití propustnosti probíhal pomocí časů provedení jednotlivých výpočetních kroků. Výpočet pro jedno zařízení je přiložen v příloze D. Tak jak je výpočet zobrazen v této příloze byl proveden pro všechna testovaná zařízení.

Výpočetní výkon zařízení, jak již bylo popsáno v kapitole (3.1), se udává v jednotkách počtu aritmetických operací v plovoucí řádkové čárce za sekundu. Pro potřeby měření je tedy nutné znát počet vykonaných aritmetických operací v rámci každého kroku. Díky přítomnosti větvení v každém kroku nelze předem stanovit přesný počet operací. Pro potřeby měření byl tedy opět použit průměr hodnot. Při testování aplikace na jednom vláknu procesoru *Intel Sandy Bridge E5-2665*, byl zaznamenán počet aritmetických operací pomocí knihovny *Performance Application Programming Interface* [11]. Tento údaj poslouží jako výchozí hodnota počtu operací v průběhu celé simulace v jednotlivých krocích. Z důvodu, že simulace byla spuštěna na všech zařízeních se stejnými počty částic je pak možné tento údaj použít pro potřeby výpočtu vyžití výpočetního výkonu. Výchozí hodnoty pro výpočty výkonu jsou k nahlédnutí v příloze E.

5.2 Porovnání na různých zařízeních

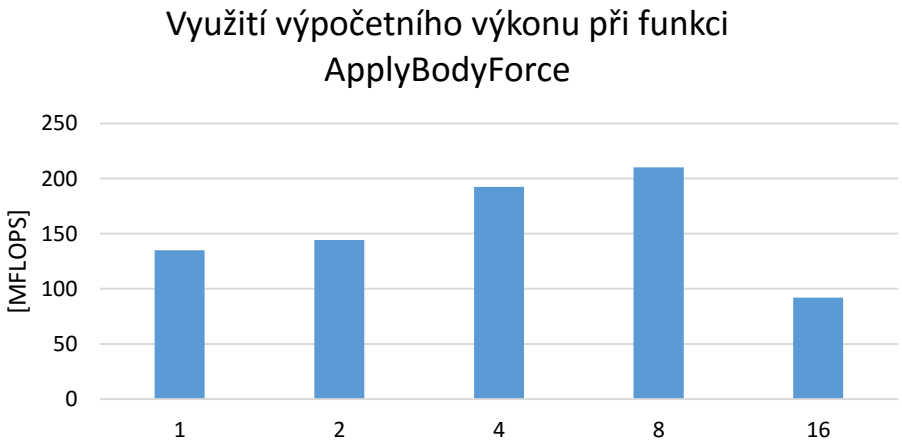
Implementace praktické části této bakalářské práce probíhala na notebooku *MSI GE62 6QC Apache*. Toto zařízení disponuje procesorem *i7-6700HQ* od firmy Intel a grafickou kartou *GTX 960M* od společnosti NVIDIA. I když se jedná o zařízení zaměřené na podání většího výpočetního výkonu, jsou na něm stále provedeny značné kompromisy pro zvýšení výdrže baterie a snížení velikosti celého zařízení. Pro zvýšení počtu různých měření bylo provedeno měření na jednom uzlu superpočítače *Anselm* obsahujícím dvojici procesorů *Intel Sandy Bridge E5-2665*.Dále pak také na plnohodnotné verzi grafické karty *HD 7970 (Tahity)* pro stolní počítač od firmy AMD. K jednotlivým zařízením je možné shlédnout jejich výkonové specifikace v příloze A měřené programem *clPeak* [12]. Hodnoty výkonu v této práci nejsou záměrně porovnávány s hodnotami uváděnými výrobcem. Primárně z důvodu, že stejně jako běžný spotřebitel osobního automobilu téměř nikdy nedosáhne spotřeby paliva garantované výrobcem, tak hodnoty výkonu ve finálním zapojení počítače nedosahují výrobcem uváděné hodnoty. Příkladem může být propustnost paměti grafické karty *GTX 960M*, kde výrobce garantuje rychlost 80 *GB/s*, viz příloha B. Oproti tomu propustnost experimentálně naměřená programem *clPeak* je o 10 *GB/s* nižší.

[15] Giga Byte za sekundu



Graf 5.1 Demonstrace snížení výpočetního času přidáním více výpočetních jader

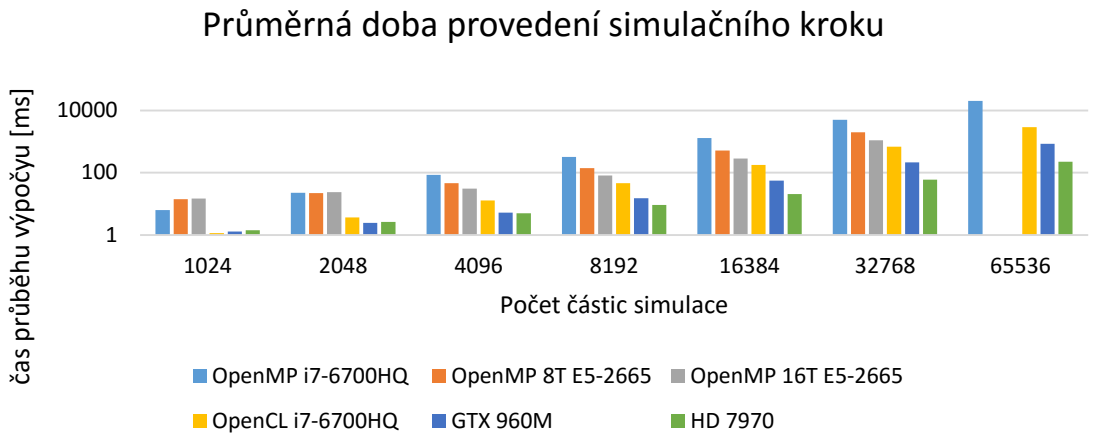
Graf 5.1 demonstruje urychlení výpočtu simulace přidáváním výpočetních jader na procesoru pomocí technologie *OpenMP*. Procesor obsahuje fyzicky čtyři výpočetní jádra, kde každé z nich je rozděleno na dvě virtuální. Tato technologie mapování jader se nazývá *Hyper-Threading* od firmy intel. Poslední sloupec pak znázorňuje dobu provedení stejné simulace na stejném procesoru pomocí technologie *OpenCL*. Z grafu pro zvětšení přehlednosti byly vypuštěny časy provedení funkcí, jejíž hodnoty se nijak významně nepodílejí na délce průběhu simulačního kroku.



Graf 5.2 Ukázka nevýhody přehnané paralelizace

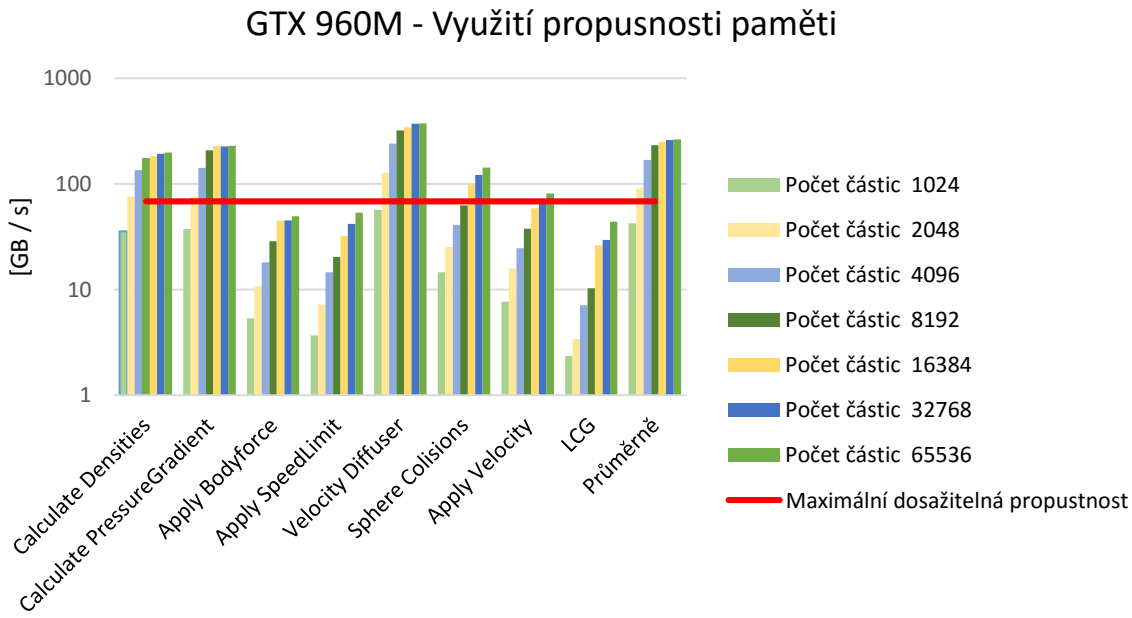
Na grafu (Graf 5.2) je možné demonstrovat, že zavádění paralelizace za každou cenu nemusí nutně znamenat zvýšení využití výpočetního potenciálu procesoru. Data zobrazená na grafu jsou čerpána z měření provedeném na uzlu počítače *Anselm* při malém počtu částic v simulačním systému.

Na grafu je patrné, že při rozdělení práce mezi dva procesory, dojde i k rozdělení uložených dat mezi ně. V důsledku rozdělení dat je pak při malém počtu částic znát zpoždění generované komunikací mezi jednotlivými procesory uzlu.



Graf 5.3Výsledky měření na testovaných zařízeních

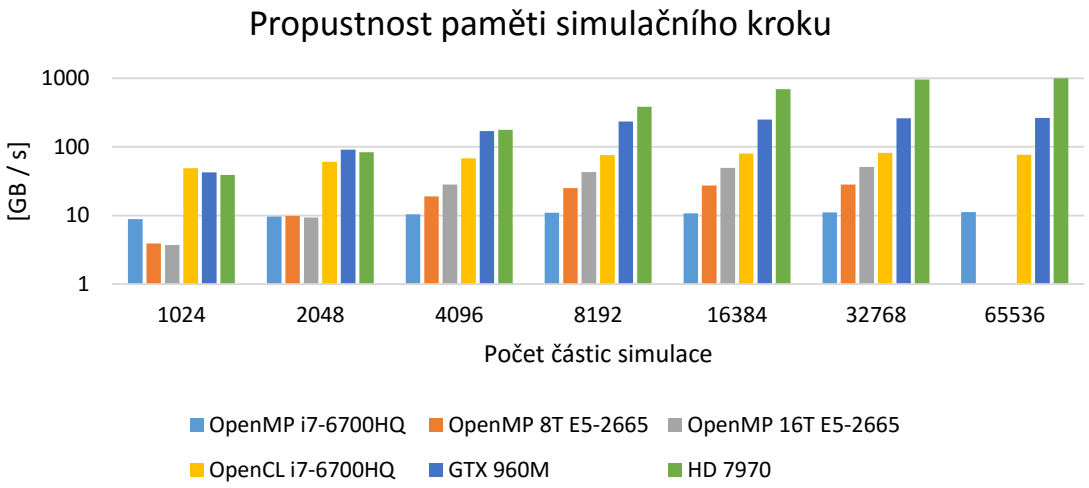
Graf 5.3 reprezentuje vizualizaci výsledů měření na použitých zařízeních. Na grafu je možné sledovat trend prodlužování doby simulačního kroku v závislosti na počtu částic v simulaci. Dále je z grafu patrný rozdíl výpočetního výkonu jednotlivých zřízení.



Graf 5.4 Výsledky výpočtu paměťové propustnosti pro konkrétní zařízení

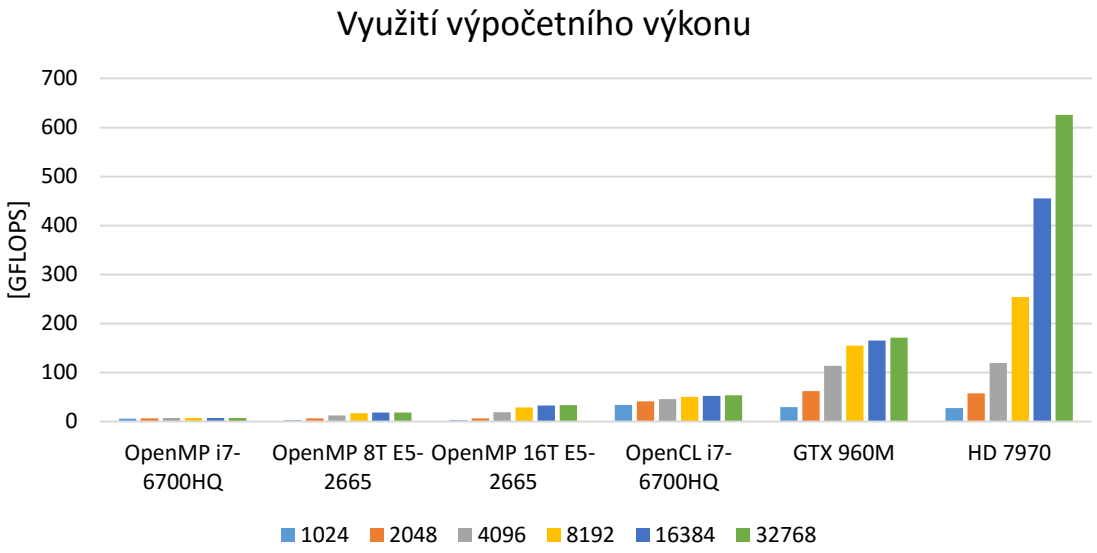
Na grafu (Graf 5.4) je zobrazeno využití propustnosti paměti na grafické kartě *GTX 960M*. Na tomto grafu je červenou spojnicí vyznačena maximální dosažitelná rychlost přenosu dat, která byla získána pomocí programu *clPeak*. K překročení maximální propustnosti paměti dochází v důsledku použití rychlé vyrovnávací paměti, která na tomto zařízení má pravděpodobně rychlost čtyřikrát vyšší. K využití jejího plného potenciálu dochází v okamžiku, kdy všechna vlákna čtou stejná data, takže

k načtení hodnoty z paměti pomalejší musí dojít pouze jednou, a ta je následně předána všem vláknům jejím prostřednictvím paměti vyrovnávací. K tomuto jevu dochází hlavně ve smyčkách, kde je nutné pro potřeby výpočtu načítat postupně informace o všech částicích aby mohlo dojít k provedení výpočtu mezi dojicí. Při takovýchto výpočetních krocích se procentuální úspěšnost nalezení hodnoty v rychlé vyrovnávací paměti pohybuje v rozmezí od 80 do 90 procent. K získání těchto hodnot byl použit program *CodeXL*, který je schopen provést měření a analýzu *OpenCL* programů puštěných na jejich zařízeních. Oproti tomu kernely jednoduché, které aplikují změny pouze na jednu částici, dosahují rychlostí pomalejších v důsledku, že každé vlákno potřebuje data rozdílná. Sledováním spojnice trendu rychlostí, je patrné že, při počtu 65536 částic již dochází k plnému zaplnění rychlé vyrovnávací paměti, přidáním částic do simulace by tedy vedlo k nižším hodnotám propustnosti. Z hlediska optimálního vytížení paměťové propustnosti lze vyčíst, že ideální počet částic v simulaci pro toto zařízení je 8192.



Graf 5.5 Výstup výpočtu dosažených přenosových rychlostí v průběhu simulace

Graf 5.5 představuje výsledky výpočtů propustností pamětí jednotlivých zařízení. Porovnáním výsledných hodnot je možné konstatovat, že podobně jako je tomu u detailního grafu (Graf 5.4) u všech použitých zařízení využívajících technologii *OpenCL* dochází k navýšení přenosové rychlosti pomocí rychlé vyrovnávací paměti. Z hlediska využití potenciálu rychlostí paměťových prvků jednotlivých zařízení lze říci, že implementace modelu SPH využívá efektivně poskytnuté možnosti jednotlivých zařízení.



Graf 5.6 Výstup výpočtu teoretického využití výpočetního výkonu jednotlivých zařízení

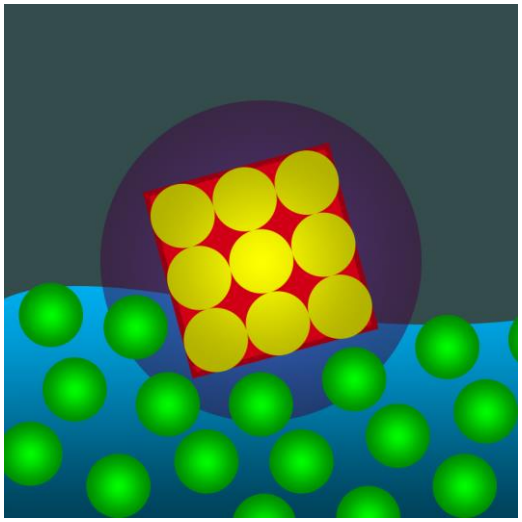
Na grafu (Graf 5.6) lze pozorovat výsledky výpočtu využití výpočetního výkonu jednotlivých zařízení v průběhu jednoho simulačního kroku. Z grafu lze vyčíst, že u velkého množství částic v simulaci se projevuje výhoda akcelerování výpočtu na grafických kartách. Hodnoty provedených operací v plovoucí řádové čárce za jednu sekundu se ale bohužel neblíží ani k 50 % využití celkového potenciálu zařízení. To je zapříčiněno tím, že v rámci porovnání každé dvojice částic v simulaci dochází k čtení paměti pro získání pozic. Tyto paměťové operace pak brzdí běh celého výpočtu, i přes úctyhodné rychlosti, kterých dosahují paměťové prvky jednotlivých zařízení. Z grafu je možné ověřit, že grafická karta *HD 7970* opravdu poskytuje čtyřnásobný výkon oproti mobilní grafické kartě *GTX 960M*.

6 Možnosti dalšího rozšíření

V této kapitole jsou nastíněny možnosti vylepšení implementovaného modelu z hlediska jeho rozšíření a dalšího použití.

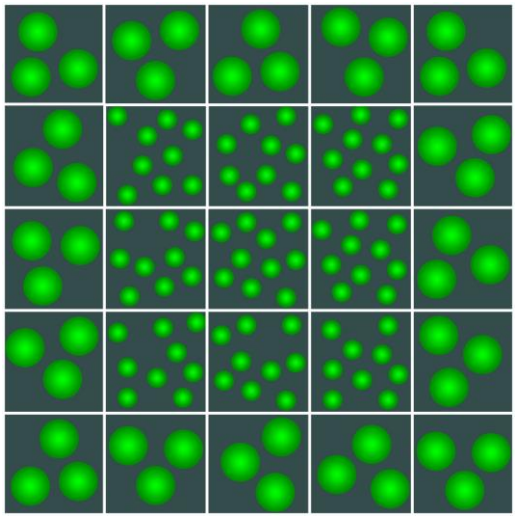
Řešení použité pro vizualizaci hladiny v implementaci není příliš vhodné pro větší simulační plochy, jelikož kubická mřížka musí být definována všude tam, kam by se mohla simulovaná kapalina teoreticky dostat. Proto by pro vizualizaci hladiny bylo vhodnější uvážit nějakou z metod pro zobrazení mračna bodů, nebo použití metody *sprite particles* pro vytvoření hladiny.

Výhodou modelu SPH je jeho částicový původ a fakt, že model dokáže zohlednit váhu jednotlivých částic. Z tohoto důvodu by model SPH mohl být vhodným kandidátem na použití v komplexnější simulaci založené na částicích. Veškeré interakce v takové simulaci by pak probíhaly mezi částicemi rozdílných vlastností. A to by umožnilo nejen detekovat kolize kapaliny s jinými tělesy, ale i vytvářet zpětnou reakci, jako například posouvání vlivem proudění, či nadnášení.



Obrázek 6.1 Vizualizace zavedeni částicové simulace

Obrázek 6.1 ilustruje možnou podobu takovéto částicové simulace, kde modrou barvou je znázorněna hladina kapaliny, jež je tvořena zelenými částicemi. V kapalině je ponořeno červené pevné těleso, které má ve své struktuře pevně umístěny částice žluté. Výhodou kulatých částic je, že lze s nimi vyplnit těleso libovolných rozměrů. Rozložení takových částic v tělese pak určuje fyzikální vlastnosti jako například polohu těžiště a přetáčení v důsledku vnějších vlivů na jeho částice. Proto, aby nemuselo docházet ke kontrole kolizí a interakcí mezi všemi částicemi komplexního systému, je by vhodné jednotlivé objekty obalit do koulí. V případě, kdy by byly koule dvou nebo více objektů v kolizi, by následně docházelo k jednotlivému vyhodnocování kolize.



Obrázek 6.2 Vizualizace zavedeni rozdílných velikostí částic

I když bylo dosaženo urychlení výpočtů simulace pomocí paralelizace výpočtu, stále není možné na dnešním hardware provést simulaci velkého množství částic v reálném čase. Za pomocí metody SPH není tedy například proveditelné simulovat protržení vodní nádrže v reálném čase. K dosažení zvětšení objemu simulované kapaliny by mohlo napomoci měnit velikosti částic a s tím i jejich váhy. Velikosti částic by bylo možné měnit na dvou principech, buď jako je tomu běžné v počítačové grafice odvozovat velikosti podle vzdálenosti pozorovatele[16], nebo částice zmenšovat tam, kde to bude mít zásadní vliv na podobu kapaliny. Takto by se dali větší částice tříštit při kolizích s objekty, nebo místech rozdílných proudů. V klidovém stavu kapaliny by bylo možné částice opět spojovat do větších.

[16] Princip známý pod názvem *Level of detail*

7 Závěr

Tato bakalářská práce se zabývá simulací proudění kapaliny v reálném čase ve virtuálním prostředí. Vysvětluje základní principy a metody jak dosáhnout vizuálně věrohodného chování kapaliny v interakci se svým prostředím. Výsledkem práce je demonstrační aplikace napsaná v jazyce *C++*, který využívá k urychlení výpočtů technologie *OpenCL* a *OpenMP*. Demonstrační aplikace umožňuje sledovat vizuální výstup simulace v reálném čase. V rámci běhu simulace je možné provádět interakci uživatele s modelem kapaliny, který na patřičné změny v simulačním systému reaguje.

Změření výsledné implementace vyplývá, že paralelizací výpočtu simulace dochází k urychlení jednotlivých kroků výpočtu simulace. Měření běhu implementace vykazuje, že hlavním problémem při běhu simulace je práce s pamětí, jelikož je v průběhu jednoho simulačního kroku nutné porovnávat všechny kombinace dvojic částic. K omezení tohoto jevu by mohlo dojít k zavedením akceleračních datových struktur, nebo jiným podobným způsobem omezit počet matematických výpočtů. Pro optimalizaci práce s pamětí by bylo také možné uvážit, zda by nebylo vhodné, aby jednotlivá výpočetní vlákna mohla zpracovávat více informací najednou, nebo si načítat větší konzistentní bloky paměti a ty pak následně zpracovávat.

V závěru této práce je tedy možné říci, že použitím technologií určených k paralelnímu zpracování výpočtu lze úspěšně akcelerovat výpočet modelu kapaliny SPH. Samotné použití těchto technologií *OpenCL* a *OpenMP* samo osobě neznamená, že okamžitě dojde k maximálnímu využití výpočetního potenciálu libovolného zařízení. Jedná se o technologie nízko úrovňové, při jejich použití je nutné brát ohled na cílový hardware. Nejedná se tedy o technologie kouzelné, ale o mocný nástroj, jehož použitím v rukou zkušeného programátora, může vniknout program, který skutečně využije plného potenciálu výpočetního zařízení.

Literatura

[1] PELÁNEK, Radek. *Modelování a simulace komplexních systémů: jak lépe porozumět světu*. Brno: Masarykova univerzita, 2011. ISBN 978-80-210-5318-2.

[2] DRÁBKOVÁ, Sylva. *Mechanika tekutin*. Ostrava: Vysoká škola báňská - Technická univerzita, 2007. ISBN 978-80-248-1508-4.

[3] SUSSMAN, M., K.M. SMITH, M.Y. HUSSAINI, M. OHTA a R. ZHI-WEI. A sharp interface method for incompressible two-phase flows. *Journal of Computational Physics* [online]. 2007, **221**(2), 469-505 [cit. 2016-05-17]. DOI: 10.1016/j.jcp.2006.06.020. ISSN 00219991. Dostupné z: http://linkinghub.elsevier.com/retrieve/pii/S0021999106002981

[4] INGEMAR KINNMARK. *The Shallow Water Wave Equations Formulation, Analysis and Application*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1986. ISBN 9783642826467.

[5] MONAGHAN, J J. Smoothed particle hydrodynamics. *Reports on Progress in Physics* [online]. 2005, **68**(8), 1703-1759 [cit. 2016-05-04]. DOI: 10.1088/0034-4885/68/8/R01. ISSN 0034-4885. Dostupné z: http://stacks.iop.org/0034-4885/68/i=8/a=R01?key=crossref.c562820df517a049ca7f11d0aefe49b4

[6] CLAVET, Simon, Philippe BEAUDOIN a Pierre POULIN. Particle-based viscoelastic fluid simulation. In: *Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation - SCA '05* [online]. New York, New York, USA: ACM Press, 2005, s. 219- [cit. 2016-05-04]. DOI: 10.1145/1073368.1073400. ISBN 176952270X. Dostupné z: http://portal.acm.org/citation.cfm?doid=1073368.1073400

[7] CHANDRAKASAN, A.P., M. POTKONJAK, R. MEHRA, J. RABAEY a R.W. BRODERSEN. Optimizing power using transformations. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* [online]. **14**(1), 12-31 [cit. 2016-05-06]. DOI: 10.1109/43.363126. ISSN 02780070. Dostupné z: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=363126

[8] *OpenMP Application Programming Interface* [online]. Listopad 2015 [cit. 2016-04-05]. Dostupné z: http://www.openmp.org/mp-documents/openmp-4.5.pdf

[9] DEHNEN, Walter a Hossam ALY. Improving convergence in smoothed particle hydrodynamics simulations without pairing instability. *Monthly Notices of the Royal Astronomical Society* [online]. 2012, **425**(2), 1068-1082 [cit. 2016-05-08]. DOI: 10.1111/j.1365-2966.2012.21439.x. ISSN 00358711. Dostupné z: http://mnras.oxfordjournals.org/cgi/doi/10.1111/j.1365-2966.2012.21439.x

[10] KARL ENTACHER. *A Collection of Selected Pseudorandom Number Generators with Linear Structures* [online]. Srpen 1997 [cit. 2016-05-08]. Dostupné z: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.53.3686

[11] *Performance Application Programming Interface* [online]. [cit. 16.5.2016]. Dostupné z: http://icl.cs.utk.edu/papi/

[12] *Clpeak* [online]. [cit. 16.5.2016]. Dostupné z: https://github.com/krrishnarraj/clpeak

[13] *Rozložení částic v kapalině* [online]. [cit. 16.5.2016]. Dostupné z: http://www.vedanasbavi.cz/upload/images/orisky/E5_2.png

[14] *Brždění pohybu částic* [online]. [cit. 16.5.2016]. Dostupné z: https://software.intel.com/sites/default/files/fluidpart15-figure-104_1.png

[15] *Schéma více jádrového procesoru* [online]. [cit. 16.5.2016]. Dostupné z: https://www.youtube.com/watch?v=cMWGeJyrc9w

[16] *Ilustrace grafického řetězce* [online]. [cit. 16.5.2016]. Dostupné z: http://return1.net/media/blogimages/opengl-points/gl-pipeline.png

[17] *Rozložení 2-dimenzinálního pracovního prostoru* [online]. [cit. 16.5.2016]. Dostupné z: https://software.intel.com/sites/landingpage/opencl/optimization-guide/OG_files/Basic_Concepts.jpg

[18] PETER SHIRLEY; R. KEITH MORLEY. *Realistic ray tracing. Peter Shirley, R. Keith Morley*. 2nd ed. Wellesley: Ak Peters, 2009. ISBN 9781568814612.

[19] MUNSHI, Aaftab. *OpenCL programming guide*. Upper Saddle River, NJ: Addison-Wesley, c2012. ISBN 0321749642.

[20] WRIGHT, Richard S., Graham SELLERS a Nicholas HAEMEL. *OpenGL superbible: comprehensive tutorial and reference*. Seventh edition. New York: Addison-Wesley, 2015. ISBN 0672337479.

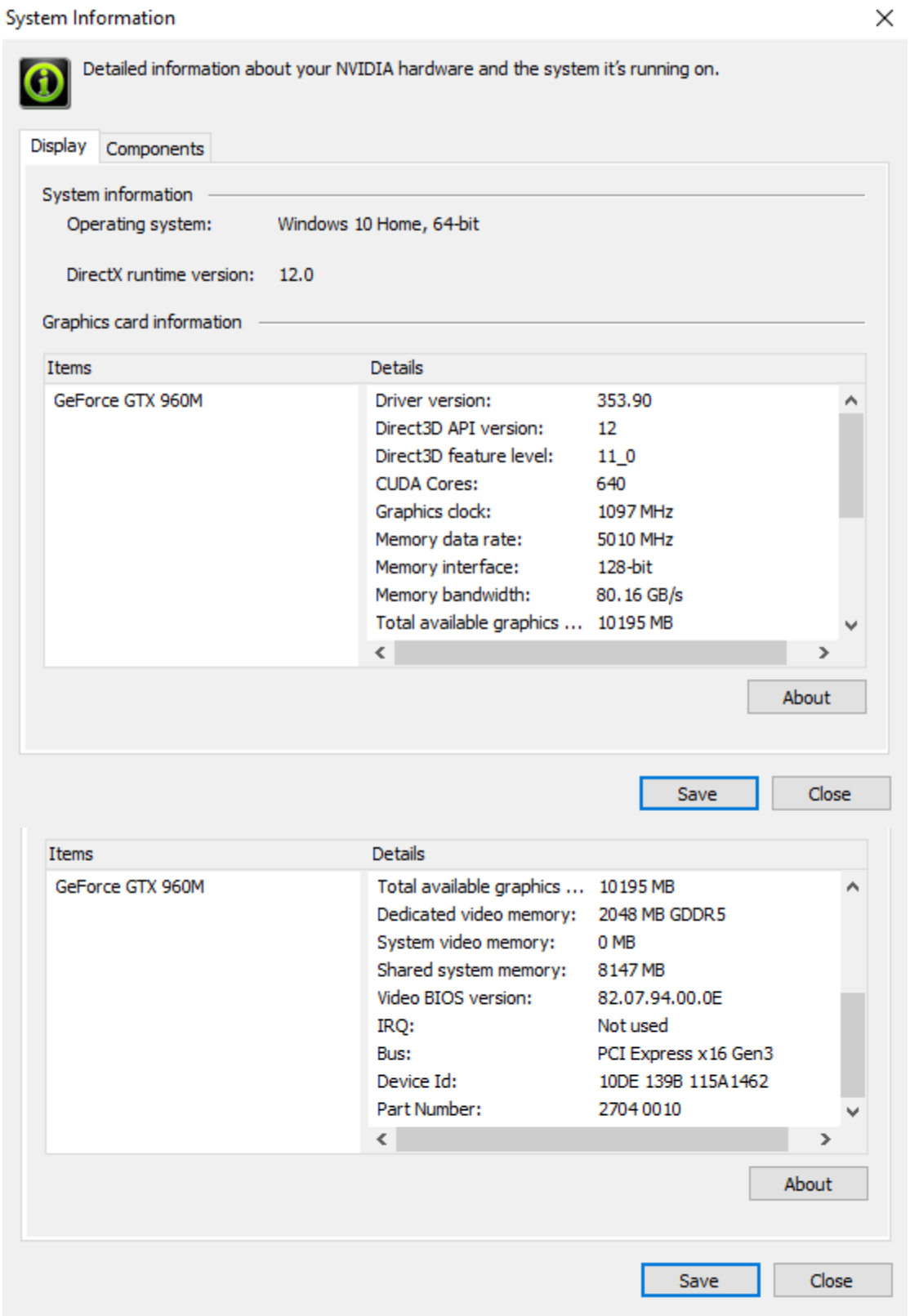
Přílohy

Seznam příloh

A Výstup programu clpeak

	i7-6700HQ	GTX 960M	HD 7970	R9 NANO
Global memory bandwidth (GBPS)				
float	22.87	68.58	238.56	447.38
float2	23.77	70.56	247.07	447.38
float4	23.87	72.05	232.13	419.42
float8	23.48	70.47	131.79	289.25
float16	23.14	70.38	66.94	113.74
Single-precision compute (GFLOPS)				
float	98.7	943.28	3817.28	7779.12
float2	100.14	1360.71	3703.25	7509.90
float4	102.39	1462.11	3654.96	7482.64
float8	100.73	1373.74	3611.17	7362.38
float16	107.84	1378.53	3605.42	7271.49
Double-precision compute (GFLOPS)				
double	49.67	---	810.54	511.59
double2	50.16	---	811.31	511.59
double4	51.52	---	810.97	509.00
double8	52.84	---	793.67	508.00
double16	59.72	---	784.04	503.53
Transfer bandwidth (GBPS)				
enqueueWriteBuffer	2.38	5.41	9.13	4.51
enqueueReadBuffer	9.19	5.2	5.61	4.02
enqueueMapBuffer(for read)	212.5	10.45	18512.79	---
memcpy from mapped ptr	9.48	8.18	4.22	3.74
enqueueUnmap(after write)	47093.94	11.93	40826.69	----
memcpy to mapped ptr	9.52	8.25	4.44	3.71
Kernel launch latency us	3.62	34.79	35.82	63.74

B Údaje o NVIDIA GeForce GTX 960M



C Ukázka výstupů profilování implementace

```
Currently used device info:
      Platform name: NVIDIA CUDA
      Device type: CL_DEVICE_TYPE_GPU
      Device name: GeForce GTX 960M
```

Simulation distance test	
DistTest1	7.41776 ms
DistTest2	16.97782 ms
DistTest3	7.42448 ms
DistTest4	6.46624 ms

Simulation parametrs	
Simulation duration	5.30481 s
Number of simulation steps	250
Number of particles	10240
Number of spheres	5
Number of grid cubes	4000
Average kernel execution time	
Linear congruential generator	0.01538 ms
Calculate densities	5.93729 ms
Calculate presure gradient	8.43295 ms
Apply body forces	0.01077 ms
Apply speed limit	0.00898 ms
Velocity difuser	6.44512 ms
Sphere colisions	0.01579 ms
Apply velocity	0.00953 ms
Cubes occupations	0.00000 ms
Average kernel execution time sum	20.87582 ms
Average sim step time	21.21906 ms

D Tabulka výpočtu přenosové rychlosti GTX 960M

			Calculate Densities	Calculate PressureGradient	Apply Bodyforce	Apply SpeedLimit	Velocity Diffuser	Sphere Colisions	Apply Velocity	LCG
čtení		float / int32 byte	12288	20480	0	0	12288	72	12	4
		float3 byte	0	12	24	12	12288	24	12	0
zápis		float / int32 buter	8	4	0	0	0	12	12	4
		float3 byte	0	12	12	12	12	12	12	12
celkem byte			12591104	21000192	36864	24576	25178112	122880	49152	20480
Počet koulí	5	**ms**	0.32991	0.52364	0.00642	0.00622	0.41062	0.00783	0.00596	0.0081
Počet částic	1024	**GB/s**	35.54417688	37.34999786	5.347706447	3.679772282	57.10619086	14.61569834	7.680598521	2.354751399
čtení		float / int32 byte	24576	40960	0	0	24576	72	12	4
		float3 byte	0	12	24	12	24576	24	12	0
zápis		float / int32 buter	8	4	0	0	0	12	12	4
		float3 byte	0	12	12	12	12	12	12	12
celkem byte			50348032	83943424	73728	49152	100687872	245760	98304	40960
Počet koulí	5	**ms**	0.61759	1.03086	0.0064	0.00635	0.73156	0.00906	0.00578	0.01122
Počet částic	2048	**GB/s**	75.92457583	75.83804373	10.72883606	7.208876722	128.1820878	25.2628958	15.83957342	3.399908436
čtení		float / int32 byte	49152	81920	0	0	49152	72	12	4
		float3 byte	0	12	24	12	49152	24	12	0
zápis		float / int32 buter	8	4	0	0	0	12	12	4
		float3 byte	0	12	12	12	12	12	12	12
celkem byte			201359360	335659008	147456	98304	402702336	491520	196608	81920
Počet koulí	5	**ms**	1.38991	2.19684	0.0076	0.00628	1.55205	0.01113	0.00743	0.01069
Počet částic	4096	**GB/s**	134.9227774	142.2983975	18.06961863	14.57846089	241.6454215	41.12881149	24.64407386	7.136945305
čtení		float / int32 byte	98304	163840	0	0	98304	72	12	4
		float3 byte	0	12	24	12	98304	24	12	0
zápis		float / int32 buter	8	4	0	0	0	12	12	4
		float3 byte	0	12	12	12	12	12	12	12
celkem byte			805371904	1342406656	294912	196608	1610711040	983040	393216	163840
Počet koulí	5	**ms**	4.27243	6.01397	0.00956	0.00895	4.66156	0.01455	0.00971	0.01483
Počet částic	8192	**GB/s**	175.5584141	207.8849118	28.72993757	20.45871159	321.8003314	62.92284149	37.71482364	10.28913625

			Calculate Densities	Calculate PressureGradient	Apply Bodyforce	Apply SpeedLimit	Velocity Diffuser	Sphere Colisions	Apply Velocity	LCG
čtení		float / int32 byte	196608	327680	0	0	196608	72	12	4
		float3 byte	0	12	24	12	196608	24	12	0
zápis		float / int32 buter	8	4	0	0	0	12	12	4
		float3 byte	0	12	12	12	12	12	12	12
celkem byte			3221356544	5369167872	589824	393216	6442647552	1966080	786432	327680
Počet koulí	5	**ms**	16.48596	21.84387	0.01222	0.01131	17.37139	0.01857	0.01243	0.01167
Počet částic	16384	**GB/s**	181.9804288	228.9167279	44.95224274	32.37939324	345.4060444	98.60283724	58.92372285	26.15045255
čtení		float / int32 byte	393216	655360	0	0	393216	72	12	4
		float3 byte	0	12	24	12	393216	24	12	0
zápis		float / int32 buter	8	4	0	0	0	12	12	4
		float3 byte	0	12	12	12	12	12	12	12
celkem byte			12885164032	21475753984	1179648	786432	25770196992	3932160	1572864	655360
Počet koulí	5	**ms**	62.00964	88.33436	0.02431	0.01746	64.51918	0.03014	0.02073	0.02071
Počet částic	32768	**GB/s**	193.5222353	226.4221362	45.19262906	41.948561	371.988085	121.5032971	70.66298842	29.47134536
čtení		float / int32 byte	786432	1310720	0	0	786432	72	12	4
		float3 byte	0	12	24	12	786432	24	12	0
zápis		float / int32 buter	8	4	0	0	0	12	12	4
		float3 byte	0	12	12	12	12	12	12	12
celkem byte			51540131840	85901180928	2359296	1572864	1.0308E+11	7864320	3145728	1310720
Počet koulí	5	**ms**	242.29178	347.29098	0.04426	0.02737	256.38003	0.05124	0.0361	0.02772
Počet částic	65536	**GB/s**	198.1102631	230.3593056	49.64450124	53.52004932	374.4469974	142.939476	81.15477839	44.03690927

E Tabulka počtu provedených aritmetických operací

Data z výztupu 1 jádra E5-2665

	1024	2048	4096	8192	16384	32768
	100	100	100	100	50	10
Apply Bodyforce	1878267	3748715	7506230	15020114	14765818	5908124
Apply SpeedLimit	621557	1249123	2564861	5374265	5939745	3766144
Apply Velocity	1572985	2876991	5402097	10340321	10480979	4654086
Calculate Densities	1220854939	4794689687	18900014428	74838783673	1.48759E+11	1.1889E+11
Calculate PressureGradient	1339290247	5257413597	20681611502	81791344668	1.62492E+11	1.29681E+11
Sphere Colisions	13784890	31979965	64974490	130215375	130330790	53374516
Velocity Diffuser	1271731431	4937754318	19239128449	75632104112	1.49704E+11	1.19211E+11

Počet operací v plovoucí řádové čárce na průměrný krok simulace

	1024	2048	4096	8192	16384	32768
Apply Bodyforce	18,783	37,487	75,062	150,201	295,316	590,812
Apply SpeedLimit	6,216	12,491	25,649	53,743	118,795	376,614
Apply Velocity	15,730	28,770	54,021	103,403	209,620	465,409
Calculate Densities	12,208,549	47,946,897	189,000,144	748,387,837	2,975,179,502	11,889,023,914
Calculate PressureGradient	13,392,902	52,574,136	206,816,115	817,913,447	3,249,841,249	12,968,083,643
Sphere Colisions	137,849	319,800	649,745	1,302,154	2,606,616	5,337,452
Velocity Diffuser	12,717,314	49,377,543	192,391,284	756,321,041	2,994,087,149	11,921,101,084
Celkem	38,498,367	150,299,172	589,016,117	2,324,240,017	9,222,354,630	36,785,011,695

Využití výpočetního výkonu [GFLOPS]

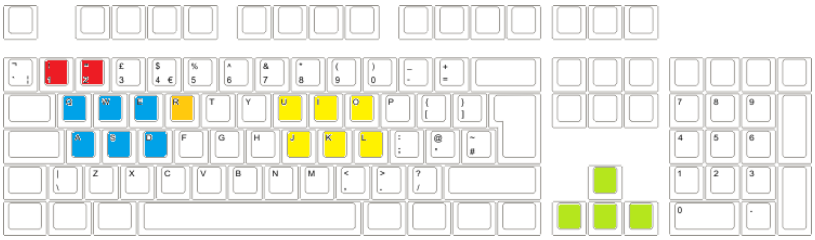
	1024	2048	4096	8192	16384	32768
OpenMP i7-6700HQ	6.197986	6.631676	6.983056	7.323522	7.100224	7.326756
OpenMP 8T E5-2665	2.734861	6.769672	12.718461	16.744411	18.092210	18.576560
OpenMP 16T E5-2665	2.594707	6.378285	19.001574	28.646224	32.689403	33.577950
OpenCL i7-6700HQ	34.122304	41.364007	45.750386	50.360189	52.600078	53.719564
GTX 960M	29.643772	62.137394	113.667324	154.891921	165.371728	171.111757
HD 7970 Tahiti	27.197332	57.481498	119.757425	254.259836	455.483627	625.760604

F Manuál

Použití souborů binárních soborů `singleCpuNoGL` a `openClNoGL` provede měření doby provedení simulace. Při zpuštění těchto soborů je možné pomocí parametrů příkazové řádky ovlivnit počet částic a kroků simulace. První celočíselný argument určuje počet částic a druhý pak počet kroků simulace.

Zpuštění binárních souborů `ibp` a `singleCpu` jsou pouštěny interaktivní dema simulace v reálném čase. Tyto soubory lze pustit s jedním celočíselným parametrem udávajícím počet částic simulace. Ovládání dema je pak následující:

- Klávesy 1, 2 slouží k přepínání mezi jednotlivými koulemi.
- Klávesy W, D, A, S slouží k pohybu zvolené koule v horizontálních osách.
- Klávesy Q, E slouží k pohybu zvolené koule ve vertikální ose.
- Kláves R slouží k obnovení výchozího stavu gravitačního pole.
- Klávesy I, J, K, L slouží k modifikaci gravitačního pole v horizontálních osách.
- Klávesy U, O slouží k modifikaci gravitačního pole ve vertikální ose.
- Šipky na klávesnici slouží k pohybu kamery po kružnici kolem simulačního prostoru.



Obrázek F.1 znázornění rozložení ovládacích prvků na klávesnici

G Obsah CD

Na CD jsou přítomny složky obsahující celistvé bloky bakalářské práce.

- Složka `bin` obsahuje přeložené spustitelné soubory pro operační systém windows, včetně potřebného adresářového systému obsahujícího soubory potřebné k spuštění jednotlivých binárních souborů, viz popis v kapitole 4.2.
- Složka `src` obsahuje zdrojové kódy vzniklé při implementaci praktické části
- Složka `vs` obsahuje soubory projektu programu VisualStudio2015 ve kterém byla provedena implementace praktické části.
- Složka `mereni` obsahuje souboru pořízené při měření výkonu aplikace na různých zařízeních a tabulky obsahující zpracovaná data.
- Složka `doc` obsahuje tuto technickou zprávu.