FAKULTA APLIKOVANÝCH VĚD
ZÁPADOČESKÉ UNIVERZITY
V PLZNI

KIV/PPR
Korelace dat akcelerometru s daty srdečního tepu

Tomáš Ott
ottt@students.zcu.cz
(A21N0062P)

12. ledna 2024

Obsah

1 Zadání

Z veřejně dostupných dat wearables zařízení ze studie zaměřující se monitorováním žen po menopauze [1] je cílem najít vzorec pro korelaci mezi 3D akcelerometrem a srdečním tepem. Vstupní časové řady normujte pomocí vláken. Pak napište jednoduchý generátor funkcí s následujícím prototypem:

```
double transform(const double acc_x, const double acc_y, const
    double acc_z);
```

Pro výslednou časovou řadu spočítejte její korelaci se srdečním tepem. Čím vyšší hodnota, tím lepší korelace. Tím vlastně dostanete fitness funkci, pokud budete generovat těla transformační funkcí evolučním/genetickým algoritmem, jako to např. dělá genetické programovaní nebo gramatická evoluce. Tento algoritmus vektorizujte.

Celý výpočet spusťte na OpenCL zařízení - tj. nemusíte psát parser výrazů, ale OpenCL driver to udělá za vás. Je vhodné, aby se výpočet míry korelace a transformace odehrál celý na GPU, čímž se pak vyhnete zbytečnému kopírování dat.

Ve finále vyberte transformační funkci s co největší korelací a tu zobrazte v grafu společně se srdeční frekvencí. Graf vygenerujte jako .svg soubor.

Samostatná práce vyžaduje využití technologií C++ vč. PSTL C++17, popř. WinAPI. Dále program musí využít buď autovektorizaci nebo manuální AVX2 vektorizaci výpočtu zadaného problému. V neposlední řadě musí i program využívat asymetrický multiprocesor - konkrétně x86 CPU a OpenCL kompatibilní GPGPU.

2 Analýza

Velký důraz by měl být v této semestrální práci kladen na důkladnou analýzu zadaného problému. Po vynaložení dostatečného úsilí v návrhu systému by mělo být možné daný problém jednoduše paralelizovat a optimalizovat, kde v případě okamžitého bezmyšlenkovitého programování je tato vlastnost dost nepravděpodobná.

2.1 Vstupní data a předzpracování

Poskytnutá data ze studie obsahují záznamy ze širokého spektra senzorů. Podle zadání je však třeba věnovat pozornost pouze akcelerometru a senzoru srdečního tepu. Data byla naměřena na 16 subjektech a soubory pro každý subjekt jsou pojmenovány prefixem `ACC` (accelerator) nebo `HR` (heart rate). Soubor obsahující záznamy ze tří os akcelerometru nabývá následujícího formátu:

```
YYYY-mm-dd HH:MM:SS.ssssss,x,y,z
```

Obdobně soubor se srdečním tepem je tvořen v následujícím formátu

```
YYYY-mm-dd HH:MM:SS,hr
```

Ikdyž se jedná o obecný formát a většina souborů tento formát respektuje, tak jeden ze souborů tento formát porušuje. Záznamy nejsou stejné délky, tudíž byl některý ze senzorů zapnut dříve či vypnut později. Při předzpracování tohoto souboru je nejdříve potřeba zahodit všechny záznamy, které nemají z druhého souboru ekvivalentní časový záznam. Seskupovat záznamy je možné pomocí vteřin, kde ale v případě dat z akcelerometru je záznamů v sekundě několik (typicky 32), tudíž musí dojít i k seskupení v tomto ohledu. Pro naše účely postačí pokud tyto hodnoty zprůměrujeme, ale pokud by se jednalo o kritickou aplikaci, tak by bylo vhodné vymyslet přístup jiný.

2.1.1 Normalizace

Součástí každého algoritmu strojového učení hledající nějaký trend v datech je i normalizace. Normalizace je proces úpravy datových hodnot na jednotnou škálu nebo standard, což umožňuje lépe porovnávat různé rysy nebo hodnoty. Cílem normalizace je zajistit, aby každý datový bod měl stejnou škálu, takže každý rys je stejně důležitý.

Min-Max Normalizace Min-max normalizace je jedním z nejčastějších způsobů normalizace dat. Pro každý rys se minimální hodnota tohoto rysu transformuje na 0, maximální hodnota se transformuje na 1 a každá jiná hodnota se transformuje na desetinné číslo mezi 0 a 1.

Například, pokud byla minimální hodnota rysu 20 a maximální hodnota byla 40, pak by se 30 transformovalo na přibližně 0,5, protože je uprostřed mezi 20 a 40. Formule je následující[3]:

$$\text{Normalizovaná hodnota} = \frac{\text{hodnota} - \min}{\max - \min}$$

Min-max normalizace má poměrně výraznou nevýhodu: velmi špatně se vypořádává s odlehlými hodnotami. Například, máte-li 99 hodnot mezi 0 a 40 a jedna hodnota je 100, pak se všechny 99 hodnoty transformují na hodnotu mezi 0 a 0,4.

Z-skóre normalizace Z-score normalizace je strategie normalizace dat, která se vyhýbá zmíněnému problému s odlehlými hodnotami. Vzoreček pro Z-score normalizaci je níže[3]:

$$\text{Normalizovaná hodnota} = \frac{\text{hodnota} - \mu}{\sigma}$$

Zde je μ střední hodnota a σ je směrodatná odchylka. Pokud je hodnota přesně rovna průměru všech hodnot rysu, bude normalizována na 0. Pokud je pod průměrem, bude záporným číslem, a pokud je nad průměrem, bude kladným číslem. Velikost těchto záporných a kladných čísel určuje směrodatná odchylka původního rysu. Pokud měla nenormalizovaná data velkou směrodatnou odchylku, normalizované hodnoty budou blíže k 0.

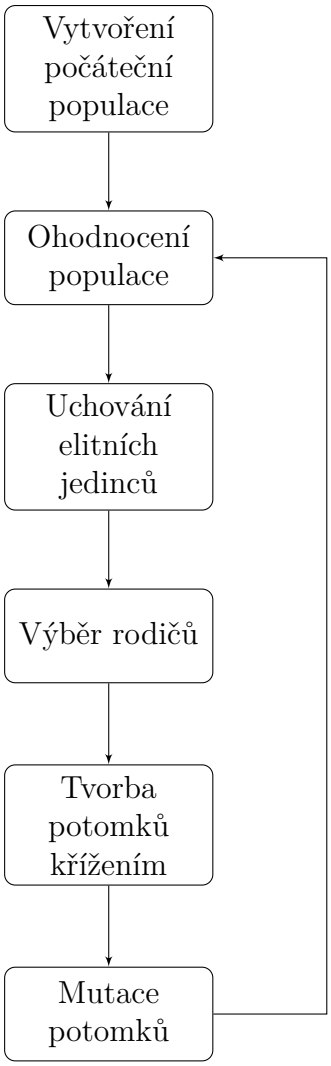
2.2 Generátor funkcí

Součástí zadání je implementovat jednoduchý generátor funkcí. Jelikož ale danou problematiku neovládáme natolik (pohled experta) nabízí se nám využití jednoho z algoritmů umělé inteligence.

2.2.1 Genetický algoritmus

Genetický algoritmus je metoda pro řešení jak omezených, tak neomezených optimalizačních problémů, která vychází z přírodního výběru, procesu řídícího biologickou evoluci. Genetický algoritmus opakovaně upravuje populaci individuálních řešení. V každém kroku genetický algoritmus vybírá jedince z aktuální populace, kteří se stanou rodiči a jsou použiti k vytvoření potomků pro následující generaci. Přes postupné generace se populace "vyvíjí"směrem

k optimálnímu řešení. Genetický algoritmus lze použít k řešení různých optimalizačních problémů, které nejsou vhodně řešitelné standardními optimalizačními algoritmy. To zahrnuje problémy, kde je objektivní funkce nespojitá, nederivovatelná, stochastická nebo silně nelineární. Genetický algoritmus může řešit i problémy kombinovaného celočíselného programování, kde jsou některé složky omezeny na celočíselné hodnoty. Následující diagram zobrazuje nejpodstatnější kroky algoritmu:



Genetický algoritmus používá tři hlavní typy pravidel při každém kroku k vytvoření následující generace z aktuální populace:

- Pravidla výběru vybírají jedince, nazývané rodiče, kteří přispívají k populaci v následující generaci. Výběr je obvykle stochastický a může záviset na skóre jedinců.

- Pravidla křížení kombinují dva rodiče a vytvářejí potomky pro následující generaci.
- Pravidla mutace aplikují náhodné změny na individuální rodiče k vytvoření potomků.
- Elitní potomek je identický s rodičem. Potomek z křížení získává části od obou rodičů. Potomek z mutace pochází od jednoho rodiče a obsahuje změnu.

2.2.2 Reprezentace populace

Genetický algoritmus vyžaduje, aby byla populace nějak matematicky reprezentovatelná. Naším cílem je najít funkci, jenž nám zvládne přetransformovat data ze tří os akcelerometru do jedné hodnoty a zjistit zda tyto hodnoty korelují s hodnotami srdečního tepu. Nabízí se nám funkce:

$$acc^T = \alpha x^i + \beta y^j + \gamma z^k + \delta \tag{1}$$

kde $\alpha, \beta, \gamma, \delta$ jsou konstanty ve specifikovaném rozsahu v oboru reálných hodnot a i,j,k jsou celočíselné kladné konstanty. Převedeme-li tuto funkci do podoby pole může vypadat následovně:

α	i	β	j	γ	k	δ

2.2.3 Fitness funkce

Fitness funkce je matematickou funkci, která hodnotí, jak dobře daný jedinec v populaci řeší daný problém. Cílem fitness funkce je kvantifikovat úspěšnost nebo kvalitu jedince na základě jeho vlastností nebo hodnot. V našem případě je fitness funkcí korelace dat transformovaného akcelerometru (využití vygenerované funkce viz sekce č.2.2.2). Korelace je statistický ukazatel, který vyjadřuje míru lineárního vztahu mezi dvěma proměnnými (což znamená, že se společně mění konstantním tempem). Je to běžný nástroj pro popis jednoduchých vztahů, aniž by vytvářel tvrzení o příčinnosti. Korelace nemůže zkoumat přítomnost nebo účinek jiných proměnných mimo ty dvě, které jsou zkoumány, tudíž potřebujeme data z tříosého akcelerometru transformovat. Vzoreček pro korelaci vypadá následovně:

$$r = \frac{n \cdot \sum (xy) - \sum x \cdot \sum y)}{\sqrt{n \cdot \sum x^2 - (\sum x)^2} \cdot \sqrt{n \cdot \sum y^2 - (\sum y)^2}} \tag{2}$$

2.2.4 Křížení rodičů

V genetických algoritmech se křížení provádí s cílem kombinovat informaci z více jedinců a vytvořit potomka, který zdědí určité vlastnosti od svých rodičů. Existuje několik způsobů křížení, které se používají v různých situacích. Mezi nejčastěji používané patří následující [4]:

1. Jednobodové křížení (Single-point Crossover)
    - Vybere se náhodný bod na genetickém řetězci rodičů.
    - Potomek zdědí část genetické informace od jednoho rodiče před bodem a část od druhého rodiče po bodu.
2. Dvoubodové křížení (Two-point Crossover)
    - Vybere se dva náhodné body na genetickém řetězci rodičů.
    - Potomek zdědí část genetické informace od jednoho rodiče mezi prvním a druhým bodem a část od druhého rodiče mimo tento interval.
3. Uniformní křížení (Uniform Crossover)
    - Každý gen v potomkovi je vybrán nezávisle na základě náhodného rozhodnutí.
    - Může být implementováno pomocí náhodné masky, která určuje, které geny se zdědí od kterého rodiče.
4. Aritmetické křížení (Arithmetic Crossover)
    - Používá se pro kontinuální proměnné.
    - Potomek je vytvořen kombinací vah rodičů, kde váhy jsou náhodně generované.

3 Implementace

Celá aplikace byla vyvíjena v C++17, sestavena pomocí cmake a Veškeré informace o sestavení aplikace jsou standardně zapsány do Cmakelists.txt souboru. V následující kapitole bude popsané jak je aplikace navržena a jakým způsobem je řešená paralelizace výpočtů.

3.1 Preprocessor

Uživatel na vstupu vkládá cestu k adresáři, který chce projít. Program tento adresář rekurzivně projde a uloží si cesty všech pod adresářů obsahující dvojici souborů ACC a HR. Po sběru těchto adresářů dojde k predikci přibližné velikosti konečných vektorů, aby nedocházelo ke zbytečně realokaci. Potřebná velikost pro jeden adresář se vypočítá následovně:

```
size_t vector_size = hr_file_size / 24;
vector_size -= vector_size % VECTOR_SIZE;
```

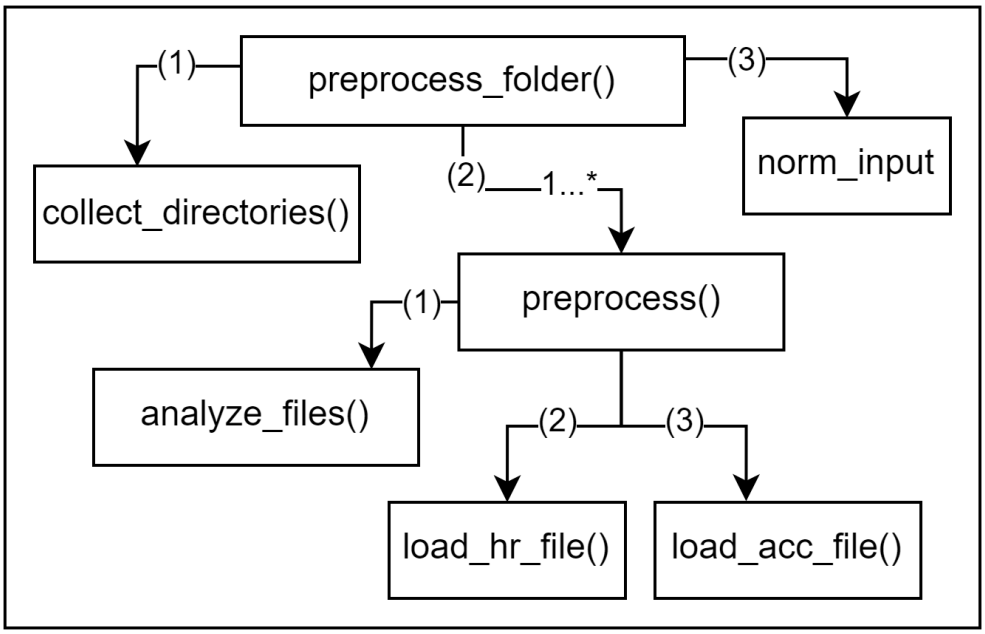
Kde hodnota 24 značí počet znaků jedné řádky v datovém souboru se srdečním tepem v nejhorším případě. Tato hodnota je pak zalomena na pevně stanovenou velikost vektoru, aby mohlo dojít k budoucí vektorizaci.

Data ze zmíněných dvojic jsou načítána a ukládána do inicializovaných `std::vector`. Jelikož soubory nemají stanovenou pevnou délku řádky, jsou načítány standardní knihovnou po řádkách. V případě vlastní implementace načítání dat by toto řešení bylo velmi neefektivní především z neustálého dotazování disku, avšak v tomto případě si knihovna data před načítá po větších kusech a ke zmiňovanému jevu nedochází. Než však dojde k samotnému načítání je potřeba o souboru zjistit důležité informace. Jedná se především o časy na prvních řádkách, tak aby bylo možné data vhodně agregovat. S datumem přichází i další problém, protože není formát unifikovaný a tak další informaci, kterou je potřeba zjistit je pozice separátoru dat a datumu. Poslední důležitá informace je kolik vzorků akcelerometru obsahuje soubor za jednu vteřinu tedy vzorkovací frekvenci.

```
time_t first_hr_time = 0;
time_t first_acc_time = 0;
uint8_t sampling_rate = 0;
size_t hr_date_end_index = 0;
size_t acc_date_end_index = 0;
```

Po zajištění těchto důležitých hodnot jsou sériově načítány řádky pomocí funkce standardní knihovny `readLine()` a ihned zpracovány do číselné

podoby. Zmíněné procesy se provádí pro všechny načtené soubory. Po dokončení načítání jsou data ještě seriově znormovány pomocí min-max normalizace. Pro odlišení od paralelní verze jsou cykly označeny pomocí `#pragma loop(no_vector)` aby nedocházelo k jejich vektorizaci. Celý tok programu při preprocessingu znázorněn na obrázku č.1.



Obrázek 1: Znázornění toku programu v Preprocessor objektu

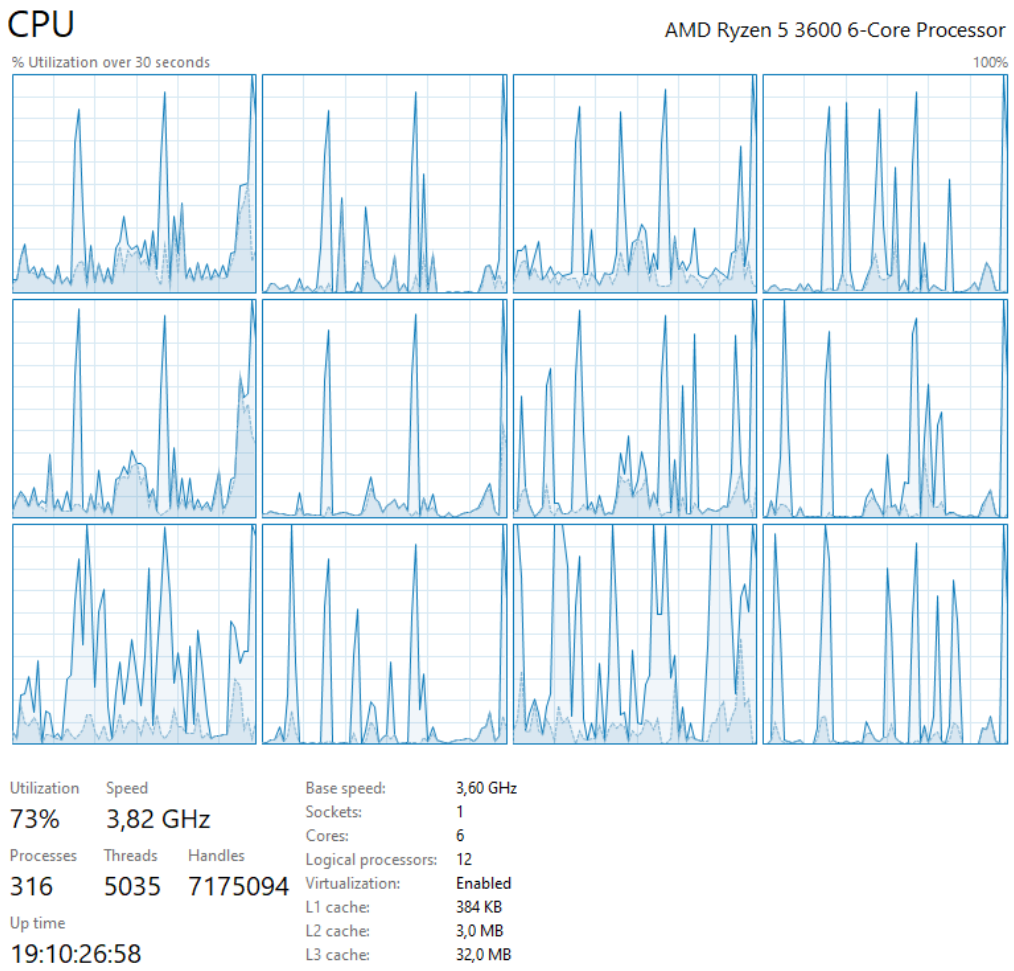
3.1.1 Paralelní preprocesor

Aby byly vyzkoušeny techniky paralelizace, bylo v objektu Preprocessoru několik metod rozšířeno o klíčové slova virtual, aby bylo možné přepsat tělo rodičovské metody tělem potomka. Jedná se o především o metody `load_acc_file_content`, `norm_input_vector` a `preprocess_acc_vectors`

V paralelní metodě `load_acc_file_content` dochází nejdříve k načtení všech dat ze souboru a pak až jejich zpracování pomocí `std::for_each` a s nastavením `std::execution::par`, aby docházelo k paralelní exekuci. Data se načítají do následující struktury:

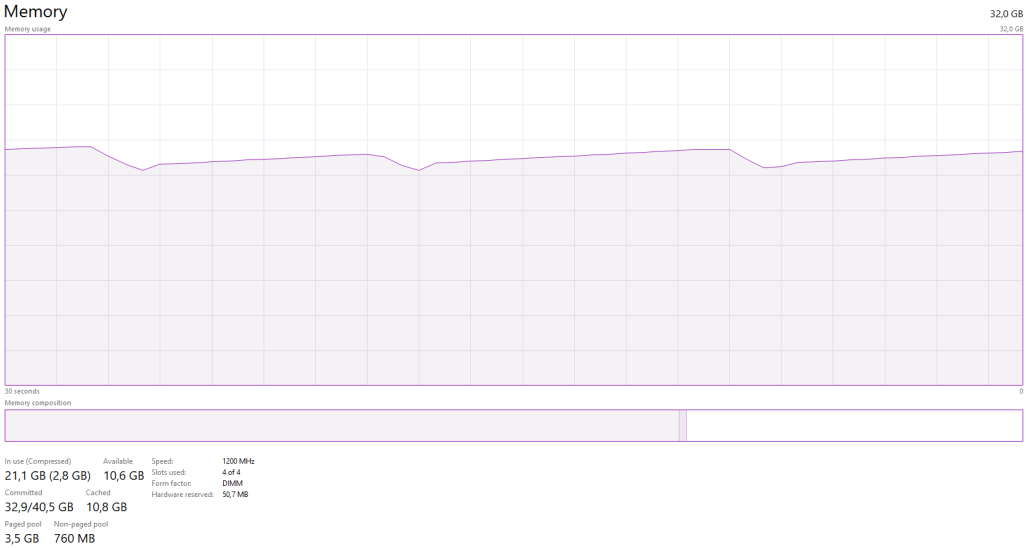
```
struct acc_minute_sample{
    size_t index{};
    std::vector<std::string> lines;
};
```

Tudíž každých n (podle vzorkovací frekvence) záznamů je agregováno do jednoho listu, který je po celém načtení souboru paralelně zpracován vlákny. Jádra procesoru jsou rovnoměrně zatížena jak je možné vidět na obrázku č.2.



Obrázek 2: Vytížení CPU při načítání souborů a předzpracování

Další zajímavý pohled je na graf vytížení operační paměti, kdy je přesně vidět kdy dochází k uvolnění paměti po zpracování jednoho souboru (obrázek č.3).



Obrázek 3: Snímek obrazovky ze správce úloh na záložce operační paměti při předzpracování souborů.

Obdobně bylo vyzkoušeno rozšíření `load_hr_file_content` metody, ale datové soubory se srdečním tepem nejsou dostatečně velké na to, aby se tento přístup amortizoval a tak zůstalo jejich načítání sériově.

Zbylé metody `norm_input_vector` a `preprocess_acc_vectors` jsou zaměřeny na využití technologie vektorizace. Pomocí další podpůrné informace je překladači (`#pragma loop(hint_parallel(VECTOR_SIZE_MACRO))`) doporučeno jakým způsobem má cykly vektorizovat. Ověřit zda došlo k vektorizaci se dá jednoduše z výpisu překladače:

```
ParallelPreprocessor.cpp(85) :info C5001: loop vectorized
ParallelPreprocessor.cpp(113) :info C5001: loop vectorized
```

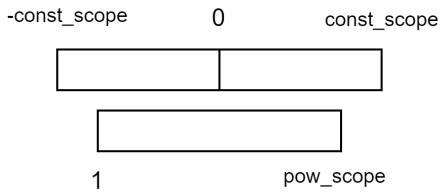
Můžeme však kromě výpisu překladače využít i funkci MSVC pro generování assembly kódu:

```
; 84   :     #pragma loop(hint_parallel(VECTOR_SIZE_MACRO))
; 85   :     for(size_t i = 0; i < count; ++i){

   test  r8, r8
   je $LN21@norm_input
   cmp   r8, 16
   jb SHORT $LN30@norm_input

$LL4@norm_input:
; 87   :         arr[i] = (value - min) / scope;

   vmovupd ymm0, YMMWORD PTR [rax-64]
   vsubpd  ymm1, ymm0, ymm5
   vdivpd  ymm2, ymm1, ymm6
   vmovupd YMMWORD PTR [rax-64], ymm2
   vmovupd ymm0, YMMWORD PTR [rax-32]
   vsubpd  ymm1, ymm0, ymm5
   vdivpd  ymm2, ymm1, ymm6
   vmovupd YMMWORD PTR [rax-32], ymm2
   vmovupd ymm0, YMMWORD PTR [rax]
   vsubpd  ymm1, ymm0, ymm5
   vdivpd  ymm2, ymm1, ymm6
   vmovupd YMMWORD PTR [rax], ymm2
   vmovupd ymm0, YMMWORD PTR [rax+32]
   vsubpd  ymm1, ymm0, ymm5
   vdivpd  ymm2, ymm1, ymm6
   add   rcx, 16
   vmovupd YMMWORD PTR [rax+32], ymm2
   lea   rax, QWORD PTR [rax+128]
   cmp   rcx, r9
   jb SHORT $LL4@norm_input
```

```
; 112 : #pragma loop(hint_parallel(VECTOR_SIZE_MACRO))
; 113 :     for(size_t i = current_offset; i < count; ++i){

   cmp   r8, rcx
   jae   SHORT $LN35@preprocess
$LN25@preprocess:
   vxorps  xmm2, xmm2, xmm2
   vcvtsi2sd xmm2, xmm2, eax
$LC15@preprocess:

; 114 :         x_input[i] /= sampling_rate;

   vmovsd  xmm0, QWORD PTR [rdx+r8*8]
   vdivsd  xmm1, xmm0, xmm2
   vmovsd  QWORD PTR [rdx+r8*8], xmm1

; 115 :         y_input[i] /= sampling_rate;

   vmovsd  xmm0, QWORD PTR [r9+r8*8]
   vdivsd  xmm1, xmm0, xmm2
   vmovsd  QWORD PTR [r9+r8*8], xmm1

; 116 :         z_input[i] /= sampling_rate;

   vmovsd  xmm0, QWORD PTR [r10+r8*8]
   vdivsd  xmm1, xmm0, xmm2
   vmovsd  QWORD PTR [r10+r8*8], xmm1
   inc   r8
   cmp   r8, rcx
   jb SHORT $LC15@preprocess
```

3.2 Calculation manager

Třída calculation manager obstarává všechny potřebné výpočty, které se týkají genetického algoritmu. Počínaje inicializací počáteční populace program generuje čísla pomocí Mersenne Twister pseudo-náhodného generátoru (který je dále používán i dalšími metodami genetického algoritmu) v rovnoměrném rozdělení. Generátor generuje čísla v rozsahu stanoveném uživatelem, kde pro konstanty se generuje v reálném rozsahu (`uniform_real_distribution`) a pro mocniny celočíselně (`std::uniform_int_distribution`) od 1 do hodnoty stanovené uživatelem (viz obrázek č.4).



Obrázek 4: Znázornění rozsahu generovaných čísel

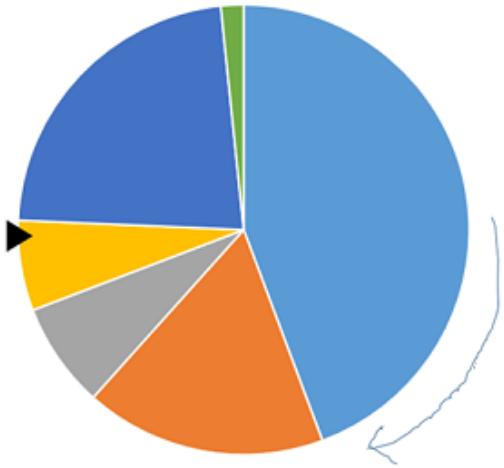
Po inicializaci následuje již hlavní cyklus algoritmu. Pomocí genomů neboli vygenerovaných funkcí představující populaci jsou transformovány data z tříosého akcelerometru. Nad transformovanou datovou sadou je vypočítána korelace s daty o srdečním tepu, jenž nám reprezentuje fitness funkci. Výsledný koeficient je uložen a na konci průchodu je z nich vybrán nejlepší. Pokud došlo k nalezení žádané korelace je simulace ukončena. Pokud ne, tak dojde k vytvoření nové populace na základně té aktuální. Celý genetický algoritmus tedy můžeme shrnout do následujících příkazů:

```
init_population();
while(step_done_count < max_step_count):
    for (const genome gen: population):
        transform(gen);
        double corr_abs = get_abs_correlation_coefficient();
        corr_result[gen_index] = corr_abs;
    if(best_correllation > desired_correllation):
        break;
    repopulate(old_population, curr_population);

    ++step_done_count;
```

Transformace tříosého akcelerometru je počítána následovným způsobem:

```
transformation_result[i] = c[0] * pow(x, p[0]) + c[1] * pow(y,
    p[1]) + c[2] * pow(z, p[2]) + c[3];
```

Repopulace Pro vytvoření nové populace po ohodnocení každého genomu korelačním koeficientem je zvolena dvojice rodičů tvořící nového potomka. Zvolení rodiče probíhá pomocí takzvaného Fitness (ruleta) úměrného výběru. To spočívá v tom, že je vygenerováno náhodné číslo určující o kolik má být "ruleta"otočena. Po otočení je zvolen rodič (viz obrázek č.5) a pokračuje se tímto způsobem dokud nedojde k repopulaci.

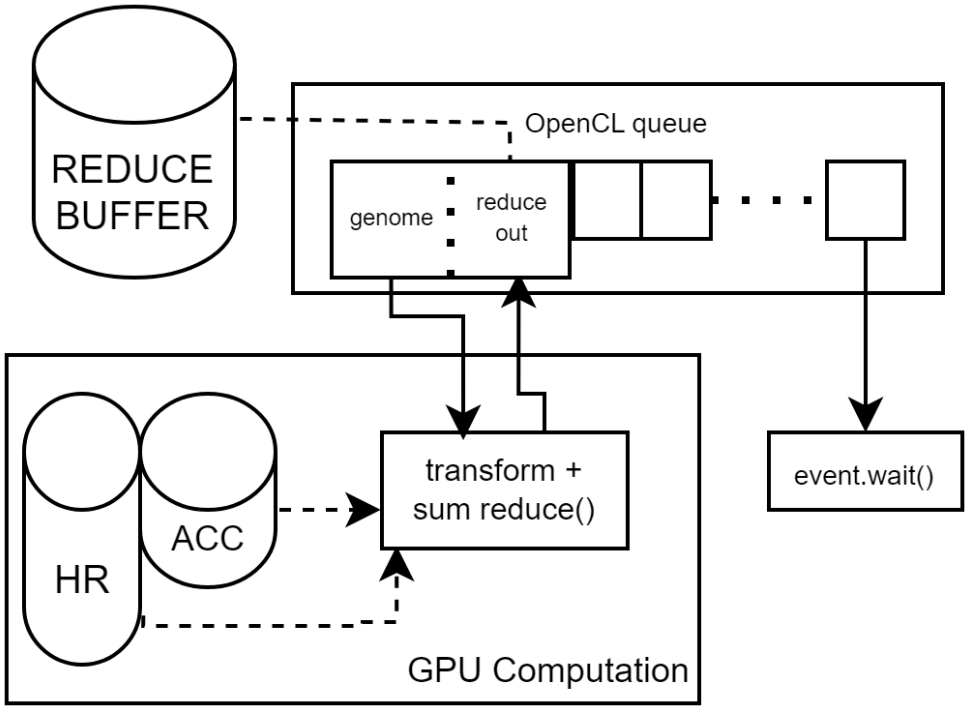


Obrázek 5: Znázornění výběru rodiče pomocí Fitness propertionate selection algoritmu.

Z jednoho z dvojice rodičů jsou předány konstanty a z druhého mocniny. Nově vzniklý genome je pak náhodně pozměněn v rámci mutace.

3.2.1 Paralelní calculation manager

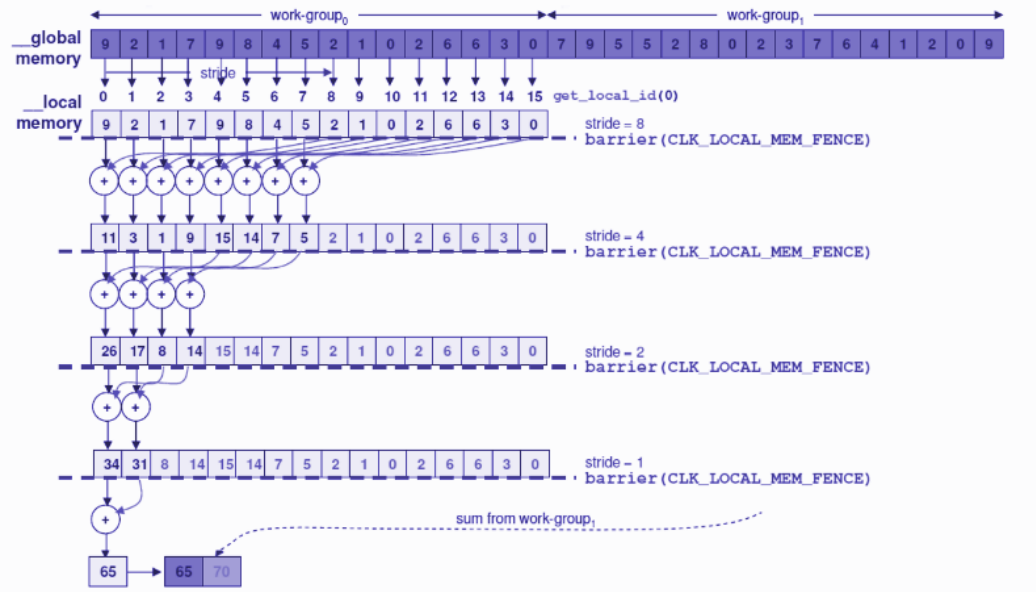
Obdobně jako je tomu u Preprocessoru i zde došlo o přetížení metody `transform_and_correlation`. Zde se používá rozhraní OpenCL pro využití výkonu grafické karty a to pro metody transformace a nově metodu redukce součtu. Pro architektonické oddělení je kód pro obsluhu OpenCL kernelu oddělen do třídy `OpenCLComponent`. Ta obstarává inicializaci statických bufferů, tedy data tříosého akcelerometru a srdečního tepu, jenž se po načtení již nemění. Tyto buffery jsou uvolněny až po kompletním ukončení genetického algoritmu. Kód je přizpůsoben tomu, aby zvládl kompletně využít kapacitu grafické karty a to díky naplnění fronty úkoly zmíněné na obrázku č.6.



Obrázek 6: Znázornění využití fronty úloh pro grafickou kartu.

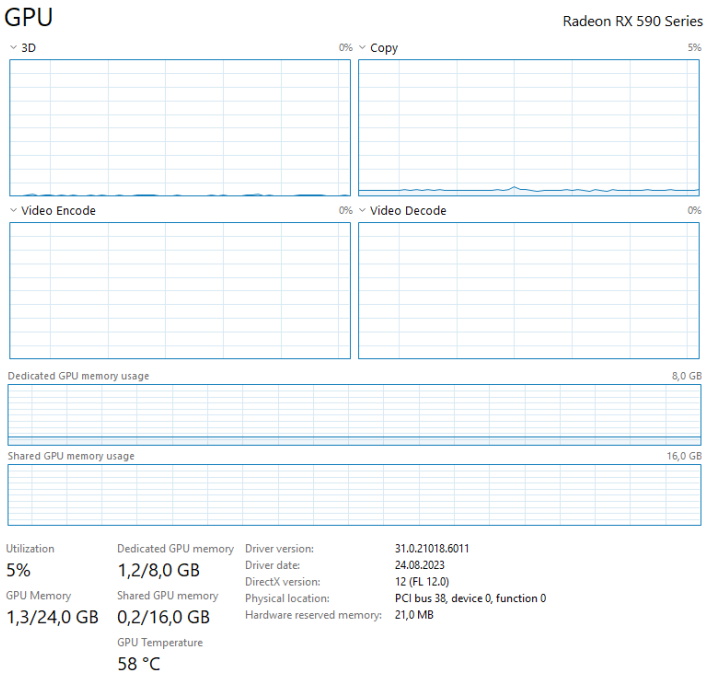
Kromě OpenCL bufferů dochází i k inicializaci bufferů pro mezisoučty sumy. Jejich velikost se během výpočtu genetického algoritmu nemění a tak zůstává v paměti po celou dobu chodu programu, jen dochází ke změně jeho hodnot. Metoda redukce součtu je oproti sériové verzi využita na grafické kartě z důvodu odlišné architektury těchto zařízení. Jedná se především o takzvané workgroupy, což je nejmenší možná skupina paralelně prováděných vláken (workitem), sdílejí paměť na grafické kartě a můžou být nezávisle

na dalších workgroup synchronizována. To umožňuje rozdělit problém do několika menších na sobě nezávislých podproblémů, jako je například metoda redukce součtu znázorněná na obrázku č.7.



Obrázek 7: Znázornění metody redukce součtu na GPU. [5]

Vytížení grafické karty bylo ze začátku měřeno pomocí správce úloh, ale při zvyšujícím se hluku a teplotě, se stále vytížení procentuálně drželo na 5% (viz. obrázek č.8). Po analýze několika výsledků google vyhledávání bylo usouzeno, že správce úloh nemusí být u všech grafických komponent přesný, jelikož pro stanovení přesného vytížení závisí na spousta faktorech. Ideální volbou je zvolení softwaru stvořeného pro daný hardware a v případě Radeon RX 590 Series se nabízí AMD Software. Jak je možné na obrázku č.9 vidět, tak grafický výpis vytížení poskytuje přijatelnější hodnoty než správce úloh. Občasné mezery mezi 100% vytížením jsou vysvětlitelné agregací mezivýsledků na CPU a zbylých kroků genetického algoritmu.



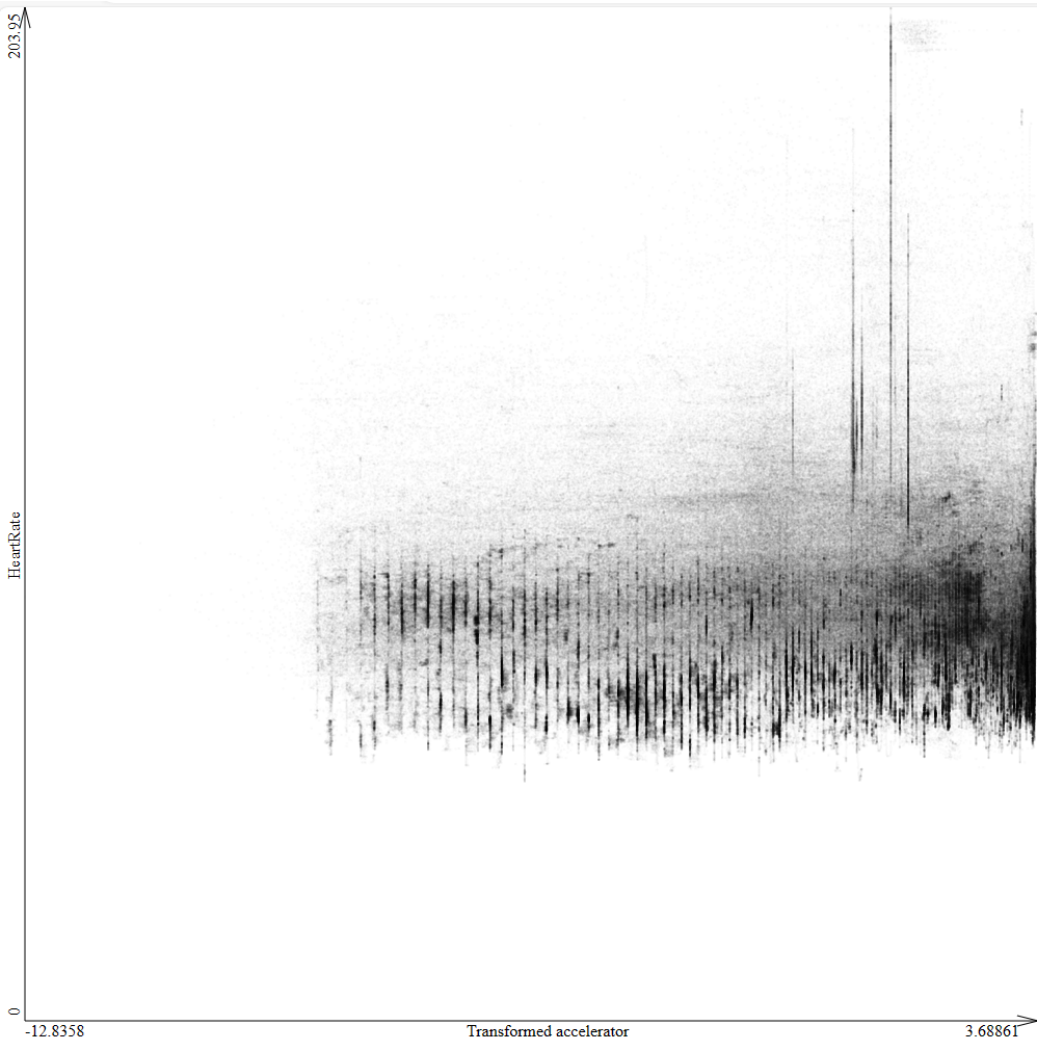
Obrázek 8: Vytížení GPU při transformaci akcelerometru podle Task Manageru

Obrázek 9: Vytížení GPU při transformaci akcelerometru podle AMD Software

3.3 Generování výstupního svg souboru

Po nalezení optimální funkce je dle zadání vygenerován svg soubor. Výsledný graf nabývá 1000x1000 bezrozměrných jednotek, obsahuje osu x znázorňující

transformovaný akcelerometr a osu y pro srdeční tep. Pro vizualizaci dat byl zvolen objekt `circle` o velikosti 1 a průhledností závislé na velikosti dat. Nejvyšší nevýhodou zvolené metody je velikost vygenerovaného souboru. Z tohoto důvodu jsou hodnoty agregovány v okolí setin a dochází tak částečnému zkreslení.



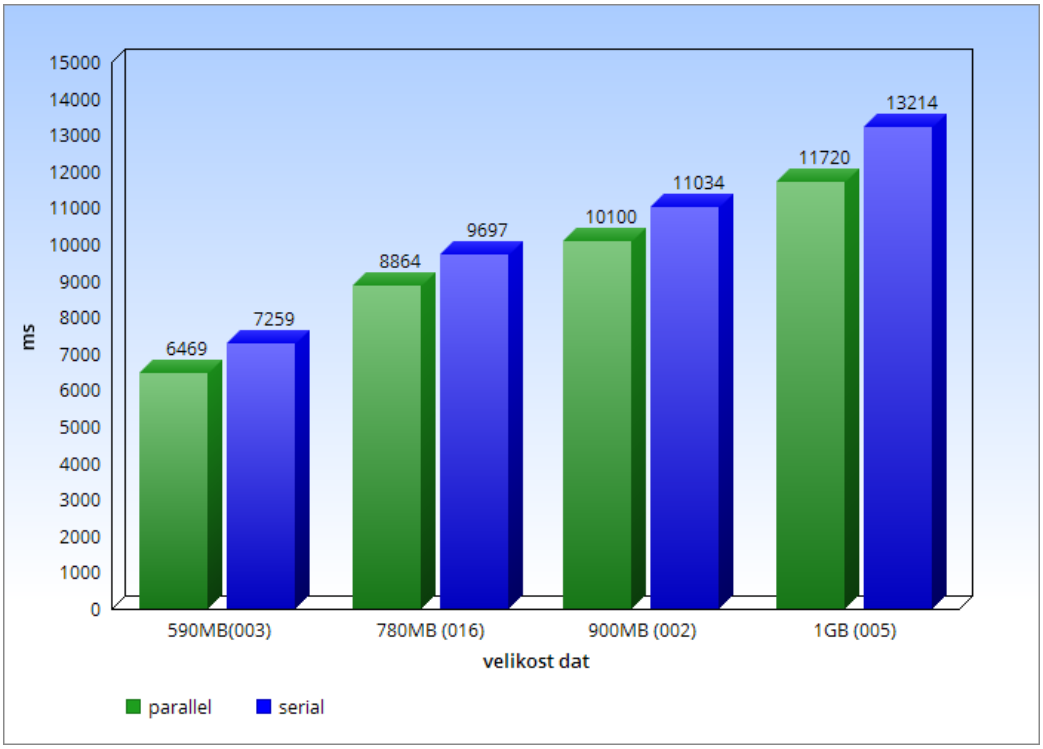
Obrázek 10: Výstup z aplikace při funkci $-4.08161x^4 + -9.68264y^2 + -2.7602z^5 + 3.68861$ a korelaci transformovaného akcelerometru $4.67639e{-}05$.

3.4 Testy výkonnosti

Bylo provedeno několik testů výkonnosti implementovaného řešení a jeho časová náročnost byla kódově naměřena. Jedná se o tři části programu, tedy tři části používající odlišné paralelizační technologie hodící se k danému typu úlohy. Testy byly prováděny na AMD Ryzen 5 3600 procesoru a Radeon RX 590 Series grafické kartě. Pro každý případ bylo naměřeno vždy nejméně 5 hodnot a došlo k jejich zprůměrování.

3.4.1 Načítání souboru

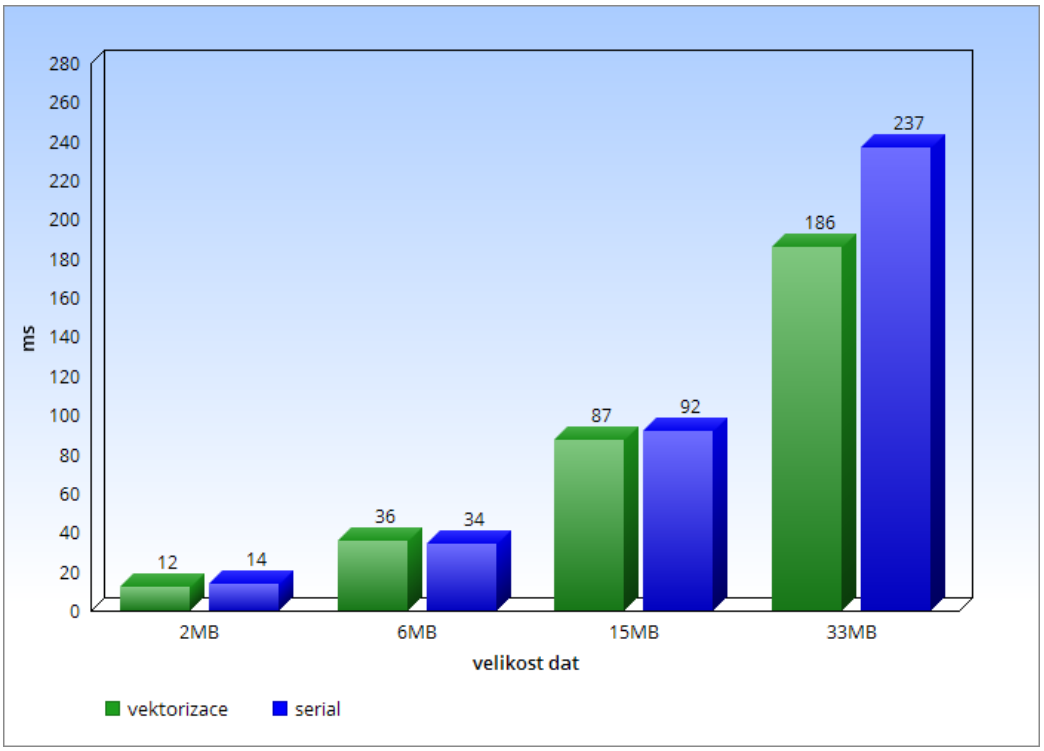
První paralelizační technologií je TBD `parallel_for_each`. Ze stanovené studie máme k dispozici několik různě velikých souborů, které se v případě načítání celé složky postupně načítají. Jak je možné spatřit na obrázku č.11, tak došlo zhruba ke zrychlení 0.1 vůči seriové verzi. Takto nízké zrychlení je způsobeno především čekáním na disk a nemožnost paralelizovat soubor do několika menších bloků (kvůli nepevné délce řádky).



Obrázek 11: Porovnání časové náročnosti načítání a zpracování jednotlivých vstupních souborů paralelním a sériovým způsobem.

3.4.2 Normalizace

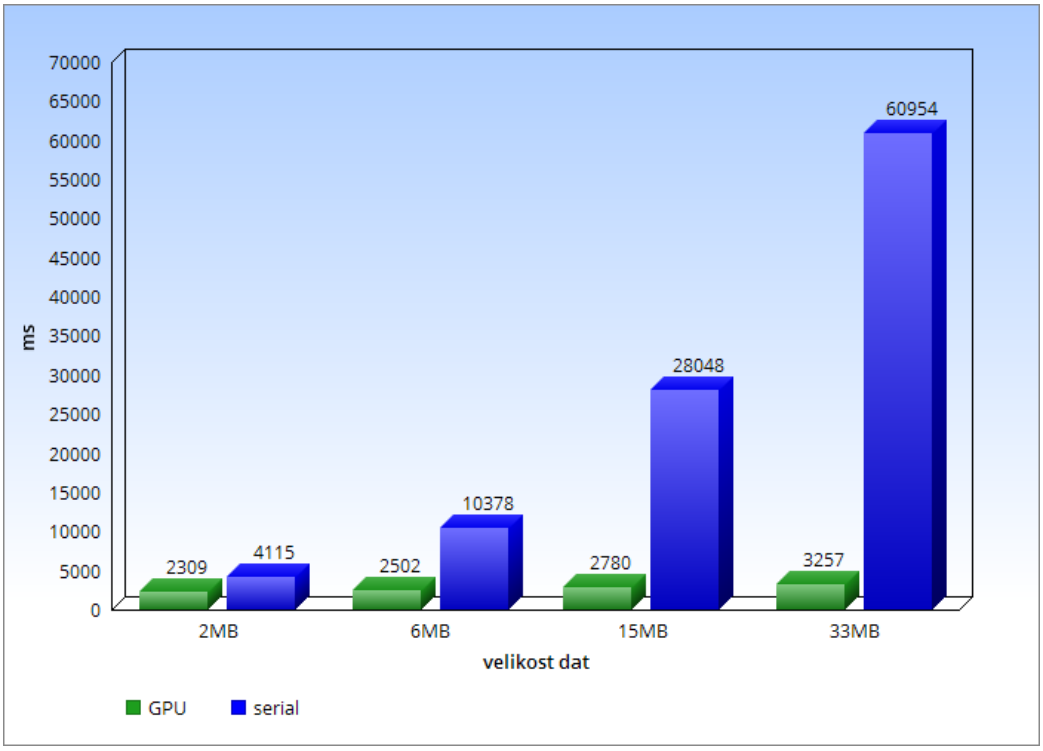
Další testovanou funkcí byla normalizace. Na obrázku č12 je znázorněno k jak velkému urychlení došlo při použití normalizace s vektorizací oproti běžnému sériovému přístupu. Velikosti referenčních dat byly stanoveny na základě počtu načtených dat celé refereční množiny (33MB), poloviny(15MB), čtvrtiny(6MB) a jedné podsložky(2MB).



Obrázek 12: Porovnání časové náročnosti normalizace všech vstupních dat pomocí vektorizace a sériového přístupu.

3.4.3 Transformace a korelace

Poslední testovanou technologií byla grafická karta. Zde došlo k neporovnatelnému zrychlení při využití výpočtu na grafické kartě oproti standardnímu sériovému přístupu. Na obrázku č.13 je zřetelně vidět, že je grafická karta výhodnější pro tento typ výpočtu. V případě, že bychom ale měli data menší než jsou na grafu znázorněny, by se mohlo stát, že bude sériový výpočet výhodnější, kvůli menší režii. Testy byly provedeny nad populací o 100 prvkách. Dalším faktorem ovlivňující rychlost výpočtu je i složitost funkce a tak není každý průchod identický.



Obrázek 13: Porovnání časové náročnosti transformace tříosého akcelerometru a výpočtu korelace na GPU se sériovým přístupem.

4 Uživatelská příručka

Program je sice díky vlastnostem nástroje cmake multiplatformní, ale byl otestován primárně na operačním systému Windows.

4.1 Závislosti

Jedinou externí závislostí jenž tento program má, je rozhraní OpenCL obalující komunikaci s grafickou kartou. [2]

4.2 Sestavení aplikace

Sestavení aplikace je díky využití cmake poměrně jednoduché. Jediné co stačí je v root adresáři zadat následující příkazy:

```
mkdir build
cd build

cmake ..
```

V případě že je použit MSVC překladač, je vygenerován visual studio solution a pak následuje příkaz ve stejné složce:

```
cmake --build . --config Release
```

Pokud se jedná o jiný překladač, měl by se vygenerovat Makefile soubor pro který by mělo stačit následující:

```
make
```

Jeden z těchto příkazů vygeneruje potřebný `ppr_ott`(.exe na Windows) soubor.

4.3 Vstupní parametry

Aplikace je rozšířená o možnost zadaní širokého spektra vstupních parametrů, jenž dokáží ovlivnit její běh. Prvním povinným parametrem je cesta k adresáři obsahující HR a ACC data. Aplikaci je tedy možné spustit následujícím způsobem:

```
C:\dev> ppr_ott.exe ./data/
```

V případě, že zadaný adresář neobsahuje žádné data. Neobsahuje dostatečně dat a nebo jiný obdobný problém, tak může být výsledný výpis následovný:

```
--------------------------------------
Running with these parameters:
Max_step_count: 10
Population_size: 100
Seed: 1704837795
Desired_correlation: 0.9
Const_scope: 10
Pow_scope: 5
Gpu_name: DEFAULT
Parallel: 0
--------------------------------------
Executing code in sequential
Loaded total of 0 directories!
Normalization of input data took 0 ms
--------------------------------------
Statistics after data load:
Entries count: 0
HR sum: 0
HR squared sum: 0
HR min/max: 1.79769e+308/-1.79769e+308
ACCX min/max: 1.79769e+308/-1.79769e+308
ACCY min/max: 1.79769e+308/-1.79769e+308
ACCZ min/max: 1.79769e+308/-1.79769e+308
--------------------------------------

Not enough data loaded! Terminating application!

Process finished with exit code 1
```

Jak bylo zmíněno dříve v textu, aplikace umožňuje zadání i jiných parametrů. Kromě parametru `-parallel` (určující že má aplikace běžet v paralelním módu), který není párový, se zbylé parametry zapisují v následujícím formátu:

```
-nazev_parametru "hodnota"
```

- `max_step_count` : Maximální počet kroků genetického algoritmu. V reálném použití by tento limit nebyl, ale pro naše účely se hodí.

- `population_size` : číslo uvádějící jak velká má být populace využívaná v genetickém algoritmu.
- `seed` : používaný seed generátorem náhodných čísel. (inicializace populace, křížení atd.)
- `desired_correlation` : Požadovaná korelace při jejimž nalezení dojde k ukončení dalšího výpočtu.
- `const_scope` : Rozsah konstant použitý při generování počáteční populace. Zadaná hodnota využívá i svůj záporný ekvivalent.
- `pow_scope` : Rozsah mocnin použitý při generování počáteční populace. Jedná se o celočíselnou kladnou hodnotu.
- `gpu_name` : Název požadované grafické karty, nebo jiného zařízení kompatibilního s rozhraním OpenCL. Zadaný název musí být přesný jinak nebude přijat.
- `step_info_interval` : Hodnota udávající jak často má docházet k vypisování průběžného nejlépe ohodnoceného výsledku genetického algoritmu.

Po zadání parametrů se změna projeví v počátečním výpisu:

```
C:\dev\ppr_ott.exe ./Testing -max_step_count "0" -gpu_name
   "Ellesmere" -step_info_interval "1" -population_size "1000"
   -parallel
--------------------------------------
Running with these parameters:
Max_step_count: 0
Population_size: 992
Seed: 1704841293
Desired_correlation: 0.9
Const_scope: 10
Pow_scope: 5
Gpu_name: Ellesmere
Parallel: 1
Input_folder: ./Testing
Step_info_interval: 1
--------------------------------------
```

5 Závěr

Program zvládá zpracovávat velké množství dostupných dat ze zvolené studie. Nad těmito daty zvládne pomocí genetického algoritmu postupně vyhledávat nejlepší lineární vztah. Tento výpočet zvládá jak sériově tak i paralelně. Po skončení výpočtu vytváří .svg soubor znázorňující vztah dat transformovaného akcelerometru a srdečního tepu. Aplikace byla otestována a optimalizovaná podle schopností autora na grafické kartě AMD Radeon RX 590 Series a později ozkoušená i na Nvidia Geforce GTX 1660 Super. Použití lineárního vztahu pro generování vztahů sice neřeší optimálně daný problém, ale aspoň to potvrzuje, že musí být k hledání optimální funkce využita jiná metoda a tím tedy splňuje zadání.

Reference

[1] Peter Cho, Juseong Kim, Brinnae Bent, Jessilyn Dunn. *BIG IDEAs Lab Glycemic Variability and Wearable Device Data* [online]. 2023 PhysioNet. Dostupné z: `https://physionet.org/content/big-ideas-glycemic-wearable/1.1.2/`

[2] Khronos Group. *OpenCL* [online]. Dostupné z: `https://www.khronos.org/opencl/`

[3] Codecademy Team. *Normalization* [online]. 2023 Codecademy. Dostupné z: `https://www.codecademy.com/article/normalization`

[4] *Crossover in Genetic Algorithm* [online]. 10 Mar, 2023 Geek for Geeks. Dostupné z: `https://www.geeksforgeeks.org/crossover-in-genetic-algorithm/`

[5] *Parallel Sum Reduction - GPU/OpenCL versus CPU* [online]. Dournac. Dostupné z: `https://dournac.org/info/gpu_sum_reduction`