



ZÁPADOČESKÁ UNIVERZITA V PLZNI

PARALELNÍ PROGRAMOVÁNÍ

KIV/PPR2

Dokumentace semestrální práce

Vojtěch DANIŠÍK
A19N0028P
danisik@students.zcu.cz

12. prosince 2021

Obsah

1	Zadání	2
2	Analýza	3
2.1	Čísla s plovoucí čárkou	3
2.2	Symetrický multiprocesor	3
2.3	GPGPU	4
3	Implementace	5
3.1	Původní řešení	5
3.1.1	Inicializace bucketů	5
3.1.2	Zpracování dat	5
3.1.3	Kontrola nalezeného bucketu	6
3.2	Nové řešení	6
3.2.1	Inicializace histogramu	6
3.2.2	Zpracování dat	7
3.2.3	Kontrola nalezeného bucketu	7
3.2.4	Control flow diagram	8
3.2.5	Data flow diagram	9
3.3	Nalezení bucketu pomocí percentilu	10
3.4	Nalezení pozice hodnoty v souboru	10
3.5	Watchdog	10
3.6	Symetrický multiprocesor	10
4	Uživatelská dokumentace	11
5	Výsledky testování	13
6	Závěr	14

1 Zadání

Program semestrální práce dostane, jako jeden z parametrů, zadaný soubor, přístupný pouze pro čtení. Bude ho interpretovat jako čísla v plovoucí čárce - 64-bitový double. Program najde číslo na arbitrárně zadaném percentilu, další z parametrů, a vypíše první a poslední pozici v souboru, na které se toto číslo nachází.

Program se bude spouštět následovně:

`pprsolver.exe soubor percentil procesor`

- soubor - cesta k souboru, může být relativní k `program.exe`, ale i absolutní
- percentil - číslo 1 - 100
- procesor - řetězec určující, na jakém procesoru a jak výpočet proběhne
 - single - jednovláknový výpočet na CPU
 - SMP - vícevláknový výpočet na CPU
 - anebo název OpenCL zařízení kompatibilní GPGPU - pozor, v systému může být několik OpenCL platforem
- Součástí programu bude watchdog vlákno, které bude hlídat správnou funkci programu

Testovaný soubor bude velký několik GB, ale paměť bude omezená na 250 MB. Zařídí validátor. Program musí skončit do 15 minut na iCore7 Skylake.

Explicitní upozornění dle postupných dotazů z řad studentů:

- Program nebude mít povoleno vytvářet soubory na disku.
- Jako čísla budete uvažovat pouze ty 8-bytové sekvence, pro které `std::fpclassify` vrátí `FP_NORMAL` nebo `FP_ZERO`. Jiné sekvence budete ignorovat. Všechny nuly jsou si stejně rovné.
- Pozice v souboru vypisujte v bytech, tj. indexováno od nuly.
- Hexadecimální číslo vypište např. pomocí `std::hexfloat`.

2 Analýza

2.1 Čísla s plovoucí čárkou

Čísla s plovoucí čárkou jsou taková čísla, která jsou moc malá nebo velká pro vyjádření v pevné řádové čárce. Čísla jsou obecně uložena jako určité množství platných číslic vynásobený exponentem. Základem exponentu bývá většinou 2, 10 nebo 16.

V případě, že bychom chtěli pomocí těchto čísel vyjadřovat intervaly bucketů v histogramu, tak je potřeba počítat s velikou časovou režii při zpracování dat. Pro všechna načtená čísla s plovoucí čárkou je potřeba zjistit do jakého bucketu bude patřit, což lze zjistit pomocí vyhledávacích algoritmů. Nejrychlejší dostupný vyhledávací algoritmus je pro nás *binární vyhledávání*, které má časovou složitost $O(\log n)$. Rychlejší řešení je reprezentace hodnot jako celočíselných indexů za pomoci *type punningu*. Type punning je programovací technika, pomocí které lze měnit datový typ proměnné, aniž bychom jakkoliv změnili její hodnotu (v bitové reprezentaci). Tímto řešením by odpadla potřeba pro každou hodnotu s plovoucí čárkou využívat binární vyhledávání, které je pro tak velké množství čísel relativně pomalé.

2.2 Symetrický multiprocessor

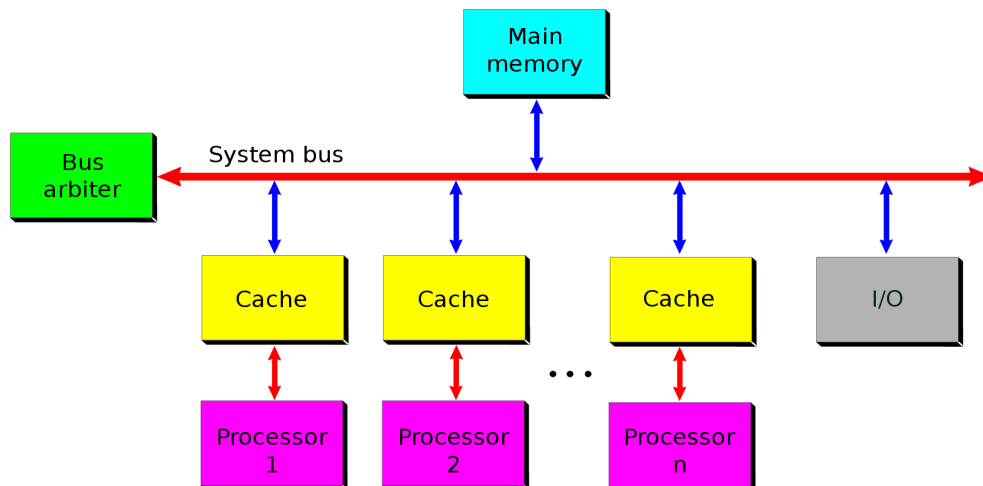
Symetrický multiprocessor (SMP) je označení pro druh víceprocesorových systémů, kde jsou dva nebo více procesorů připojeny k jedné sdílené hlavní paměti, mají plný přístup ke všem vstupním a výstupním zařízením a jsou řízeny jedinou instancí operačního systému.

Operační systém zachází se všemi procesory stejně, žádný si nevyhraduje pro speciální účely.

Každý procesor je propojen s vlastní vyrovnávací pamětí, která může být rozdělena na vyrovnávací paměť určenou pro data a vyrovnávací paměť pro instrukce). Tato vyrovnávací paměť je napojena na interní sběrnici, která zajišťuje řízení přenosu dat mezi těmito paměťmi a pamětí operačního systému (viz obrázek č. 1).

V této úloze bude SMP sloužit hlavně pro paralelní zpracování vstupního souboru, kdy hlavní vlákno může (skoro) bez omezení stále načítat datové bloky a vedlejší vlákna budou mezitím zpracovávat načtené bloky dat. V případě velmi rychlého zpracovávání dat bude nutné odladit počet vláken tak, aby nedocházelo ke zbytečnému uspávání jak vedlejších vláken, tak i hlavního vlákna.

SMP - symmetric multiprocessor system



Obrázek 1: Ukázka SMP

2.3 GPGPU

General-purpose computing on graphics processing units (neboli GP-GPU) je způsob využití paralelizace na grafické kartě k výpočtům, které jsou jinak zpracovávány pomocí *Central processing unit* (CPU).

Pomocí GPGPU pipeline lze paralelizovat proces mezi jedním až více GPU a CPU. Protože GPU operuje na nižších frekvencích, tak grafické karty mají o to více procesorů, díky čemuž lze zpracovávat více dat za sekundu.

V této úloze bude GPGPU sloužit pro zpracování bloků dat - klasifikace a rozřazení do jednotlivých bucketů. Každému vláknu lze předat část dat (ideálně vždy jedno číslo), které klasifikuje a zjištěnému bucketu přičte frekvenci. Tento přístup by zaručoval rychlejší zpracování celého souboru.

3 Implementace

V implementaci jsou popsány dva postupy řešení - původní, který nesplňoval časový limit pro single procesor, a nový, který je více časově efektivnější.

3.1 Původní řešení

Původní řešení zahrnovalo práce s čísly v plovoucí čárce - minima a maxima histogramu / bucketů, binární vyhledávání.

3.1.1 Inicializace bucketů

Inicializace bucketů a jejich minimálních a maximálních povolených hodnot se provádělo pomocí vzorce č. 1:

$$step = \left\lfloor \frac{min}{bucket_count} \right\rfloor + \frac{max}{bucket_count} \quad (1)$$

kde *step* je krok pro spočítání maximální povolené hodnoty bucketu, *min* a *max* jsou minimální a maximální povolené hodnoty v histogramu, *bucket_count* je počet bucketů v histogramu.

Buckety jsou seřazeny vzestupně (první bucket má minimální hodnotu rovno minimální hodnotě histogramu, poslední bucket má maximální hodnotu rovno maximální hodnotě histogramu).

3.1.2 Zpracování dat

Pro každou iteraci zpracování se prováděly tyto kroky:

1. Načtení datového bloku o velikosti 1 MB.
2. Kontrola validity všech načtených double čísel v datovém bloku - kontrola pomocí metody `std::fpclassify`.
3. Rozřazení čísel do správných bucketů pomocí binárního vyhledávání.
 - Rozřazování probíhalo na základě porovnání načteného čísla s minimem a maximem daného bucketu.
 - Binární vyhledávání bylo řešeno rekurzivně.
 - Při rozřazování se také aktualizovala minimální a maximální načtená hodnota v daném bucketu.
4. Inkrementace čítačů načtených hodnot.

- Čítač všech načtených validních hodnot.
- Čítač pro hodnoty, které jsou validní, ale menší jak definovaná minimální hodnota histogramu.

5. Nalezení bucketu pomocí percentilu (viz sekce č. 3.3).

3.1.3 Kontrola nalezeného bucketu

Po zpracování dat a nalezení bucketu bylo potřeba zkontrolovat, zda bucket je reprezentován jako číslo nebo jako interval.

V případě, že nalezená minimální a maximální hodnota ze souboru v daném bucketu jsou stejná, pak stačí pro nalezené číslo najít jeho pozici (viz sekce 3.4) v souboru a program se ukončí.

V případě, že nalezená minimální a maximální hodnota ze souboru v daném bucketu nejsou stejná, pak je potřeba nastavit nové minimum a maximum histogramu a opakovat celý proces zpracování dat znova.

3.2 Nové řešení

Nové řešení zahrnuje práce s celými čísly (indexy), převodem hodnot s plovoucí čárkou na celá čísla, kódování záporných čísel do sign-magnitude formátu a obcházení datových typů.

Control flow diagram lze vidět na obrázku č. 2. Data flow diagram lze vidět na obrázku č. 3.

3.2.1 Inicializace histogramu

Při inicializaci histogramu bylo potřeba provést několik věcí. Jako první se vytvoří buckety s nastavenou frekvencí na hodnotu 0 a přiřadí se jim index. Následně bylo potřeba spočítat velikost jednoho bucketu (viz vzorec č. 2).

$$bucket_size = \frac{max - min}{bucket_count - 1} + x; x = \begin{cases} 1 & (max - min) \% (bucket_count - 1) > 0 \\ 0 & (max - min) \% (bucket_count - 1) = 0 \end{cases} \quad (2)$$

kde $bucket_size$ je velikost bucketu, max je maximální hodnota histogramu, min je minimální hodnota histogramu a $bucket_count$ je počet bucketů v histogramu.

Následně se bylo potřeba vypočítat offset pro bucket indexy (viz vzorec č. 3).

$$bucket_index_offset = \frac{-min}{bucket_size} \quad (3)$$

kde *bucket_index_offset* je offset pro index bucketu, *min* je minimální hodnota histogramu a *bucket_size* je vypočítaná velikost bucketu.

Velikost bucketu a offset pro index bucketu se budou používat při převodu hodnoty na index (a zpět).

3.2.2 Zpracování dat

Rozdíl ve zpracování dat oproti starému řešení spočíval v převádění hodnot s plovoucí čárkou na celočíselné indexy, což je o mnoho rychlejší jak binární vyhledávání.

Převod je realizován ve dvou krocích:

1. Uložení hodnoty s plovoucí čárkou do binární podoby.
2. Převedení hodnoty v binární podobě ze sign-magnitude formátu.

Po převodu hodnoty lze vypočítat index bucketu, do kterého dané číslo patří. Výpočet indexu lze vidět na vzorci č. 4.

$$index = \frac{value}{bucket_size} + bucket_index_offset \quad (4)$$

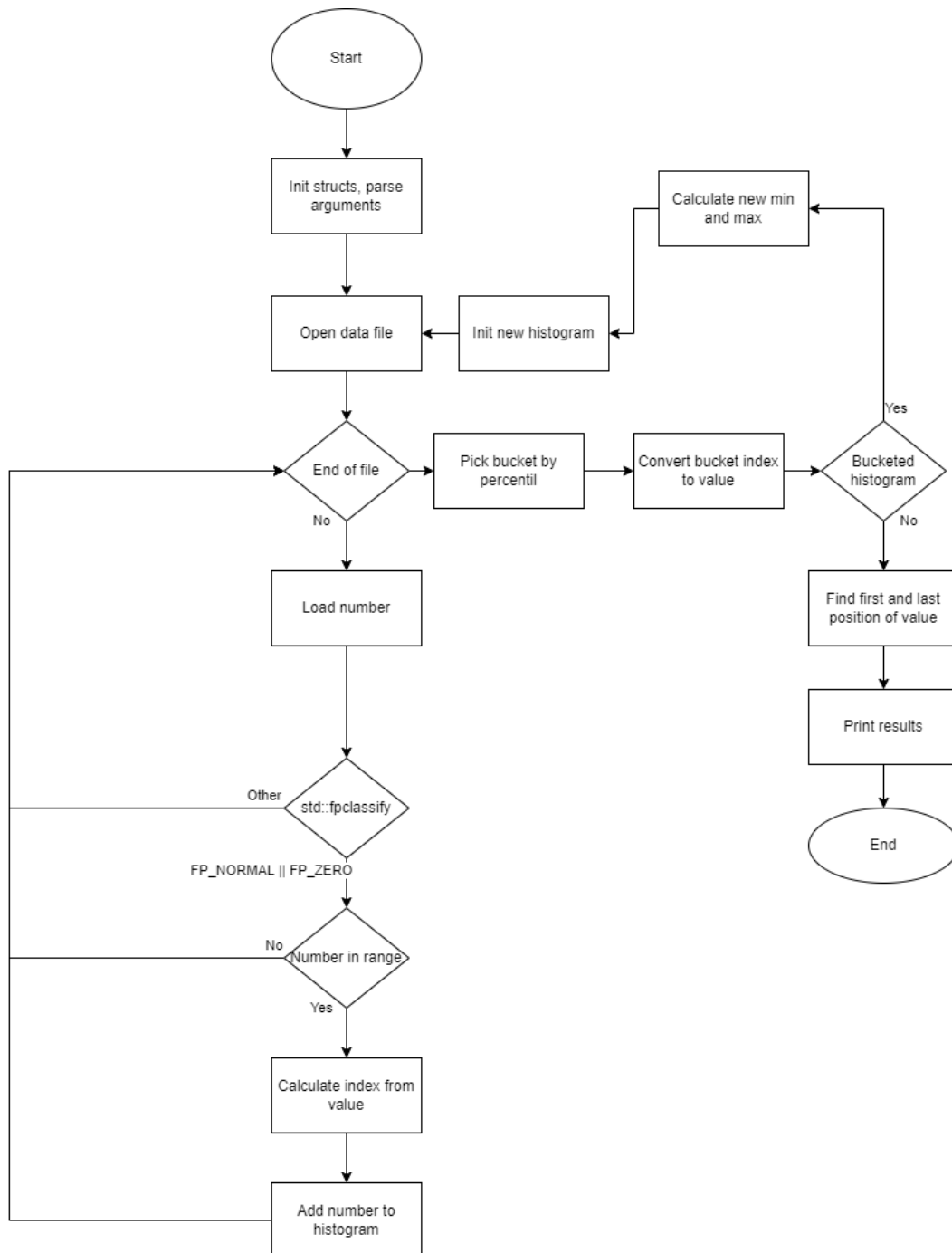
3.2.3 Kontrola nalezeného bucketu

Kontrola nalezeného bucketu u nového řešení byla jednodušší, protože se kontrolovala pouze velikost nalezeného bucketu.

Pokud velikost bucketu byla rovna jedné, tak se pouze index bucketu převedl zpět na číslo s plovoucí čárkou.

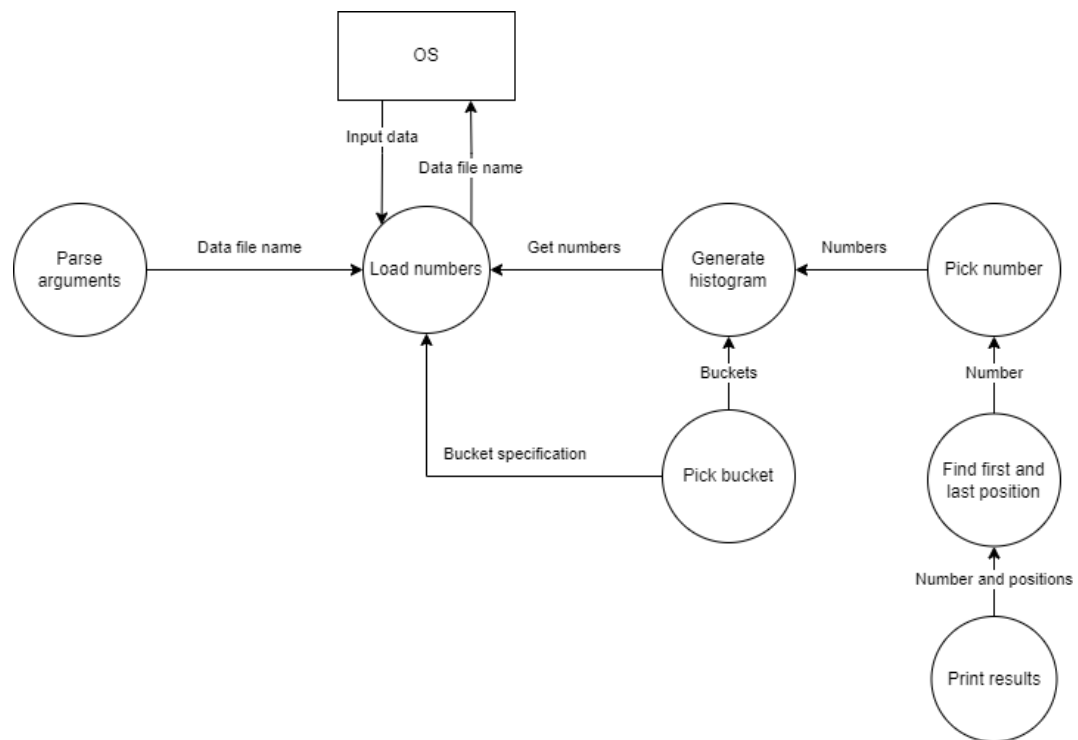
Pokud velikost bucketu byla větší jak jedna, jednalo se o interval. Index nalezeného bucketu se převedl zpět ze sign-magnitude formátu a byl určen jako minimální hodnota histogramu. Maximální hodnota histogramu byla určena jako součet minima a velikosti bucketu a celý postup se opakoval.

3.2.4 Control flow diagram



Obrázek 2: Control-flow diagram

3.2.5 Data flow diagram



Obrázek 3: Data-flow diagram

3.3 Nalezení bucketu pomocí percentilu

Po zpracování všech datových bloků v dané iteraci bylo potřeba najít výsledný bucket podle vstupního percentilu. Nejdříve bylo potřeba spočítat pozici čísla pomocí vzorečku č. 5:

$$calculated_position = \lfloor \frac{numbers_count \cdot percentile}{100} \rfloor \quad (5)$$

kde *calculated_position* je pozice hledaného čísla, *numbers_count* je počet všech hodnot a *percentile* je vstupní hodnota percentilu (v procentech).

Následně se definuje kumulativní frekvence, ke které se přičte počet hodnot, které jsou menší jak definovaná minimální hodnota histogramu. Po definici kumulativní frekvence se prochází všechny buckety a sčítá se jejich frekvence s kumulativní frekvencí do té doby, dokud kumulativní frekvence je menší nebo rovna vypočtené pozici *calculated_position*. Jakmile je kumulativní frekvence větší jak spočtená pozice, tak jsme našli náš hledaný bucket na daném percentilu.

3.4 Nalezení pozice hodnoty v souboru

Pro nalezení pozice hodnoty je potřeba postupně načíst celý soubor a porovnávat aktuální první a poslední pozici v souboru. Při inicializaci první pozice je nutné nastavit hodnotu na maximální hodnotu datového typu, zatímco pro poslední pozici je nutné nastavit hodnotu na minimální daného datového typu. Nalezené pozice se následně vynásobí osmi (pozice chceme v bytech).

3.5 Watchdog

Watchdog vlákno bylo realizováno jako jedno vlákno, které má v sobě čítač reprezentující počet uběhnutých milisekund, kdy aplikace nevykonává žádnou práci. Resetování čítače se provádí pomocí metody *reset()*, volané skoro ve všech metodách. Limit pro ukončení aplikace je nastaven na 10 sekund neaktivity.

3.6 Symetrický multiprocessor

Při použití symetrického mutliprocessoru se vytvoří 10 vláken, které mají za úkol zpracovávat datové bloky viz sekce č. 3.1.2, 3.2.2.

Předávání dat vláknům je zařízeno pomocí blokovací fronty, jejíž maximální velikost je nastavena na 40 bloků. Blokovací fronta funguje na způsob

single producent - multiple consuments.

Hlavní vlákno (single producent) načítá datové bloky a vkládá je do fronty. V případě, že fronta je zaplněná, se hlavní vlákno uspí pomocí podmínkové proměnné.

Vytvořená vlákna (multiple consuments) vybírají z fronty datové bloky a zpracovávají je. V případě, že fronta je prázdná, se vlákno uspí pomocí podmínkové proměnné.

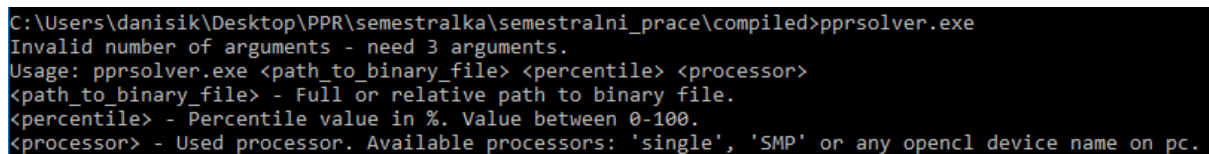
Po ukončení zpracování dat v dané iteraci je z vlákna předán zpět jeho histogram, který se následně spojí s hlavním histogramem.

4 Uživatelská dokumentace

Aplikace potřebuje ke správnému spuštění tři povinné argumenty (viz obrázek č.4):

`pprsolver.exe <path_to_file> <percentile> <processor>`

- `path_to_file` - relativní / absolutní cesta k souboru s daty
- `percentil` - číslo 1 - 100
- `procesor` - Typ procesoru, který se má použít pro výpočet:
 - `single` - jednovláknový výpočet na CPU
 - `SMP` - vícevláknový výpočet na CPU
 - název OpenCL zařízení - př. NVIDIA GeForce 940MX



```
C:\Users\danisik\Desktop\PPR\semestralka\semestralni_prace\compiled>pprsolver.exe
Invalid number of arguments - need 3 arguments.
Usage: pprsolver.exe <path_to_binary_file> <percentile> <processor>
<path_to_binary_file> - Full or relative path to binary file.
<percentile> - Percentile value in %. Value between 0-100.
<processor> - Used processor. Available processors: 'single', 'SMP' or any opengl device name on pc.
```

Obrázek 4: Výstup z konzole - argumenty programu

Výstup programu (viz obrázek č.5) je ve formátu:

`<value_in_hex> <first_position> <last_position>`

Popis hodnot:

- `value_in_hex` - nalezená hodnota na daném percentilu v hex formátu

- first_position - první pozice nalezené hodnoty v souboru v bytech
- last_position - poslední pozice nalezené hodnoty v souboru v bytech

```
C:\Users\danisik\Desktop\PPR\semestralka\semestralni_prace\compiled>pprsolver.exe party.mp3 37 smp  
-0x1.cef9cc6b375bbp-751 6821440 6821440  
C:\Users\danisik\Desktop\PPR\semestralka\semestralni_prace\compiled>pprsolver.exe party.mp3 37 "nvidia geforce 940mx"  
-0x1.cef9cc6b375bbp-751 6821440 6821440
```

Obrázek 5: Výstup z konzole - ukázka použití a výstupu programu

Pozn. Typ procesoru je case-insensitive, což znamená, že pro single procesor lze použít - single, Single, SINGLE, ...

5 Výsledky testování

Při testování byly použity 2 různě velké soubory. Rychlost výpočtu pro OpenCL u starého řešení není zmíněna z důvodu změny algoritmu.

Testovací stroj:

- CPU: Intel Core i5-8250
- Grafická karta: NVIDIA GeForce 940MX, Intel UHD Graphics 620
- Disk: HDD
- OS: Windows Home 10

Soubor: MP3; velikost: 6,8 MB

Typ procesoru	Rychlost výpočtu [s] (staré řešení)	Rychlost výpočtu [s] (nové řešení)
Single	1.1	0.1
SMP	0.4	0.2
NVIDIA	(neimplementováno)	1.5
Intel	(neimplementováno)	2.8

Tabulka 1: Porovnání rychlosti výpočtu pro soubor s velikostí 6,8 MB

Soubor: Ubuntu 20.04.3 LTS (iso); velikost: 3 GB

Typ procesoru	Rychlost výpočtu [s] (staré řešení)	Rychlost výpočtu [s] (nové řešení)
Single	950	70
SMP	375	65
NVIDIA	(neimplementováno)	105
Intel	(neimplementováno)	77

Tabulka 2: Porovnání rychlosti výpočtu pro soubor s velikostí 3 GB

Z výsledků lze vidět, že nové řešení je oproti starému řešení mnohonásobně rychlejší. Zároveň lze zmínit rychlost SMP oproti Single CPU, u které se očekávalo mnohem větší zrychlení (nejspíše zapříčiněno neideální implementací vláken / producenta-konzumenta).

6 Závěr

Výsledný program splňuje všechny požadavky zadání. Program má správně ošetřeny vstupní parametry a výsledkem programu je nalezená hodnota na daném percentilu a její pozice v souboru.

Pro soubor o velikosti 3 GB byla rychlost výpočtu v nejhorším případě (OpenCL NVIDIA) zhruba 105 sekund. Paměťová náročnost byla v nejhorším případě (OpenCL) maximálně 160 MB. Při porovnání rychlosti výpočtů starého a nového řešení se podařilo urychlit Single CPU v průměru 10x a SMP v průměru 6x, zatímco paměťová náročnost zůstala neměnná.

Z výsledků testování také vyplývá, že rychlost výpočtu SMP oproti single CPU je nepatrně rychlejší, což indikuje špatnou implementaci ať už BlockingQueue (časté čekání vláken na datové bloky), nebo práce s vlákny.