Klasifikace dat do pravděpodobnostního rozdělení

KIV/PPR - semestrální práce

Mukanova Zhanel
A22N0130P

27. ledna 2023

Obsah

1 Zadání

Zadáním semestrální práce bylo vytvořit program, který dokáže přiřadit vstupní data do jednoho ze čtyř rozdělení: Normálního/Gaussovo, Poissonovo, Exponenciálního či rovnoměrného. Program musí na konci vypočtu vypsat hodnoty charakterizující rozdělení a zdůvodnění svého výsledku. Také zadání definuje následující limity, které musí být dodržené:

- Testovaný soubor bude velký několik GB
- Paměť bude omezená na 1GB.
- Program musí skončit do 15 minut na iCore7 Skylake.

Jako vstupní parametry musí program přijímat:

- **soubor** - cesta k souboru, může být relativní k program.exe, ale i absolutní
- **procesor** - řetězce určující, na kterých procesorech výpočet proběhne, a to zároveň
  - all - použije CPU a všechny dostupné GPU
  - SMP - vícevláknový výpočet na CPU
  - **názvy OpenCL zařízení** jako samostatné argumenty - pozor, v systému může být několik OpenCL platforem

Součástí programu je také watchdog vlákno, které hlídá správnou funkci programu. Rozšířila jsem vstupní argumenty o zadání počtu vláken, které budou využité na každém CPU, výběr optimalizovaného řešení a výběr času, jak často musí watchdog kontrolovat výsledky.

2 Analýza

2.1 Původní nápad

Původně jsem počítala, že k řešení semestrální práce použiji statistické metody "Kolmogorovův–Smirnovův test" [wik22a]. Během samotné implementace jsem narazila na problém, že podobné metody typu „Goodness of fit tests" se nehodí pro odhad pravděpodobnostního rozdělení. Nelze pomocí těchto testů zjistit parametry rozdělení, které byly využity pro generování sady dat. Tento bod zadání by nebyl splněn:

> *Program vypíše hodnoty charakterizující rozdělení a zdůvodnění svého výsledku.*

Navíc, algoritmus „Kolmogorovův–Smirnovův test" vyžaduje řazení dat, což je složitá úloha pro velkou sadu dat.

2.2 Realizace

Mojí druhou myšlenkou bylo vytvořit dvouprůchodový program, který by v prvním průchodu sbíral statistiky a v druhém průchodu sestavoval histogram dat. Statistiky jsou nutné, aby se v následujících krocích spočítaly parametry rozdělení, samotný histogram a také pro nalezení chyby mezi pozorovanou a očekávanou frekvenci dat pomocí metody „Residual sum of squares".

2.3 Maximum likelihood estimation

Maximum likelihood estimation (MLE) je metoda odhadu, která nám umožňuje pomocí vzorku odhadnout parametry pravděpodobnostního rozdělení, které data vygenerovalo. Úkolem MLE je odhadnout skutečný parametr Θ který je spojen s neznámým rozdělením, pomocí kterého byl vytvořen vstupní vzorek dat. Vzorce pro každé rozdělení, které byly použité v této práci vypadají následujícím způsobem:

2.3.1 Normální rozdělení

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i$$

$$\sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2$$

$$SD = \sqrt{\sigma^2}$$

Kde μ odpovídá střední hodnotě, σ^2 rozptylu a SD směrodatné odchylce [MT21b].

2.3.2 Exponenciální rozdělení

$$\lambda = \frac{n}{\sum_{i=1}^{n} x_i}$$

Kde lambda λ je parametr exponenciálního rozdělení, který musí odpovídat $\lambda > 0$ podle definice rozdělení [MT21a].

Střední hodnotu ($Mean$) a směrodatnou odchylku (SD) pro exponenciální rozdělení se počítá následujicím způsobem:

$$Mean = min(X_1, X_2, \cdots, X_n)$$

$$SD = \frac{1}{n}\sum_{i=1}^{n} x_i - min(X_1 + X_2 + \cdots + X_n)$$

2.3.3 Poissonovo rozdělení

$$\lambda = \frac{1}{n}\sum_{i=1}^{n} x_i$$

Kde λ je střední hodnota rozdělení a musí odpovídat $\lambda > 0$ podle definici rozdělení [MT21c].

2.3.4 Rovnoměrné rozdělení

$$a = Mean = min(X_1, X_2, \cdots, X_n)$$

$$b = max(X_1, X_2, \cdots, X_n)$$

$$SD = b - a$$

Kde $X_1..X_n$ jsou náhodná data. Parametry rovnoměrného rozdělení odpovídají minimální (a) a maximální (b) hodnotám dat.

2.4 Hustota pravděpodobnosti (Probability Density Function, PDF)

Hustota pravděpodobnosti (Probability Density Function, PDF) je v teorii pravděpodobnosti funkce, jejíž integrací na kterémkoli vzorku vyjde relativní pravděpodobnost. Je-li $\rho(x)$ hustota pravděpodobnosti spojité náhodné veličiny X, pak platí:

$$\int_{\Omega} \rho(x)\mathrm{d}x = 1$$

kde Ω je definiční obor veličiny X. Pro hodnoty x mimo definiční obor Ω je hustota pravděpodobnosti nulová, takže $\rho(x) = 0$ pro $x \notin \Omega$.

Dále budou ukázany funkce hustoty pravděpodobností pro každé rozdělení, který byly vuyžity v této práci.

2.4.1 Hustota pravděpodobnosti pro normální rozdělení

$$f(x) = \frac{e^{-X^2/2}}{\sqrt{2\pi}}$$

2.4.2 Hustota pravděpodobnosti pro exponenciální rozdělení

$$f(x) = e^{-X}$$

2.4.3 Hustota pravděpodobnosti pro rovnoměrné rozdělení

$$f(x) = 1$$

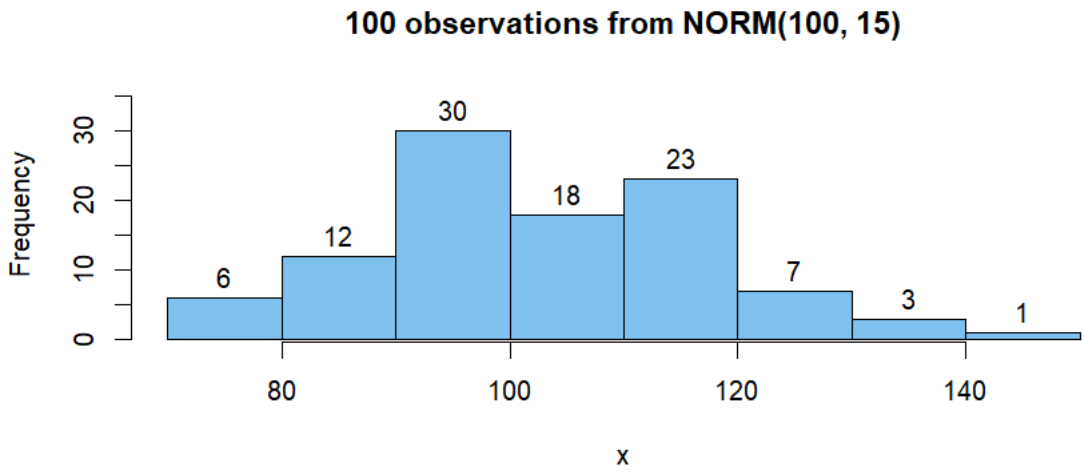
2.4.4 Hustota pravděpodobnosti pro poissonovo rozdělení

$$f(x) = \frac{e^{-\mu}\mu^X}{X!},$$

kde μ je střední hodnota Poissonovo rozdělení a X je náhodna hodnota.

2.5 Frekvenční histogram (frequency histogram)

Frekvenční histogram je grafické znázornění distribuce dat pomocí sloupcového grafu se sloupci stejné šířky, vyjadřující šířku intervalů (tříd), přičemž výška **sloupců vyjadřuje četnost** sledované veličiny v daném intervalu. Je důležité zvolit správnou šířku intervalu, neboť nesprávná šířka intervalu může snížit informační hodnotu diagramu.



Obrázek 2.1: Ukázka frekvenčního histogramu

V této práci využívám konstantní šířku histogramu, kterou pro spojitá rozdělení počítam pomocí:

$$binCount = log(n) + 2$$

$$w = (max - min)/binCount$$

a pro diskretní (poissonovo rozdělení):

$$binCount = max - min$$

$$w = 1,$$

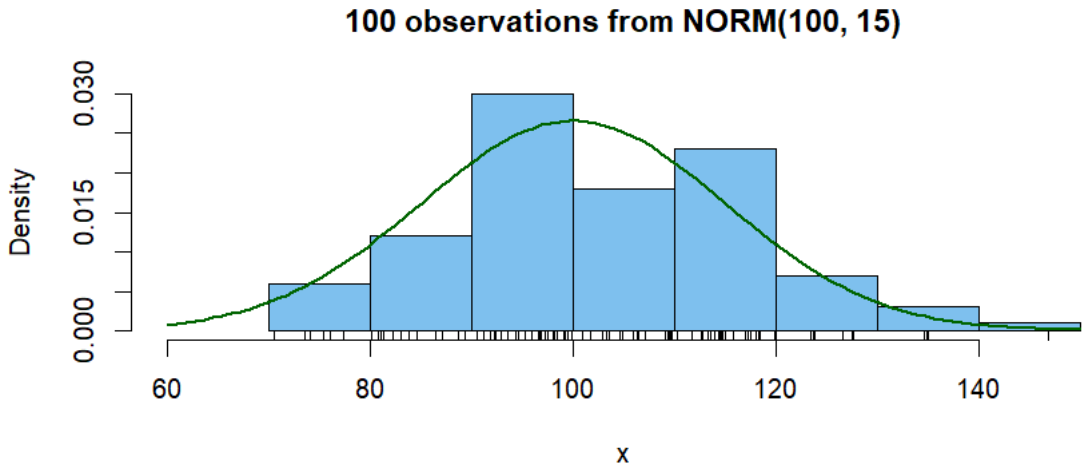
kde $binCount$ je počet sloupců v histogramu a w je šířka sloupce.

2.6 Histogram hustoty (density histogram)

Histogram hustoty se liší od frekvenčního tím, že jeho sloupce ukazují jednotky, díky nimž se suma všech hodnot sloupců je ekvivalentní 1 [Bru18]. To umožňuje porovnávat hodnoty s funkci hustoty pravděpodobnosti (PDF) rozdělení, jejíž integral se také roven 1.

Pokud f_i je frekvence i-tého sloupce, pak jeho relativní frekvence je $r_i = f_i/n$, kde n je velikost pozorovaných dat. Jeho hustota je potom $d_i = r_i/w_i$, kde w_i je jeho šířka. Vysledná formula pro převod frekvenčního histogramu na histogram hustoty je:

$$d_i = f_i/w_i/n$$



Obrázek 2.2: Ukázka histogramu hustoty

2.7 Residual sum of squares (RSS)

Ve statistice je **Residual sum of squares (RSS)** [Wik22b] součtem čtverců rozdílů mezi daty a modelem odhadu, v našem případě jde o předpokládané rozdělení. Čím je menší hodnota RSS, tím je menší rozdíl mezi pozorovanými a očekávanými daty. Tato hodnota se používá jako optimální kritérium při výběru parametrů a modelu (rozdělení pravděpodobnosti).

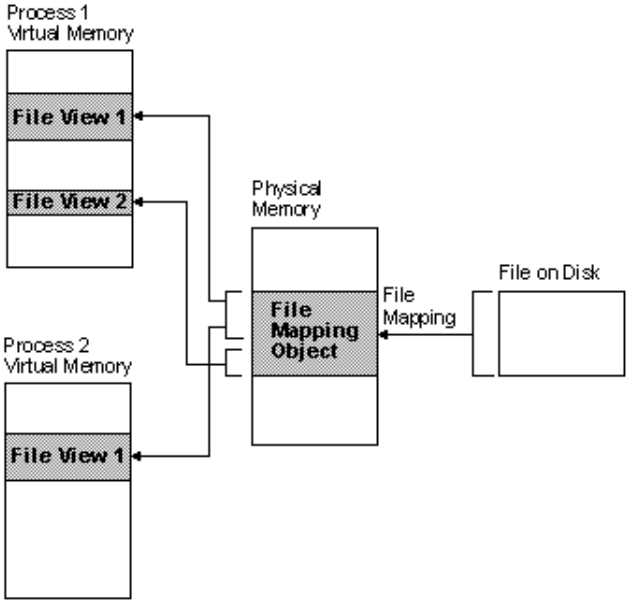
$$RSS = \sum_{i=1}^{n} (y_i - f(x_i))^2$$

Kde za y_i dosazují hodnoty d_i (viz sekci 2.6) a za $f(x_i)$ funkci hustoty pravděpodobnosti (viz sekci 2.4).

3 Implementace

3.1 File Mapping

S doporučením od pana docenta jsem rozhodla ve své práci využít mapování souboru [Mic21a] do adresního prostoru procesu místo klasického čtení ze souboru (viz obrázek 3.1). Mapování souboru umožňuje programu efektivně pracovat s velkým datovým souborem, aniž by bylo nutné mapovat celý soubor do paměti. Další výhodou **"File Mapping"** je to, že ke sdílené mapované stránce může také přistupovat více vláken.

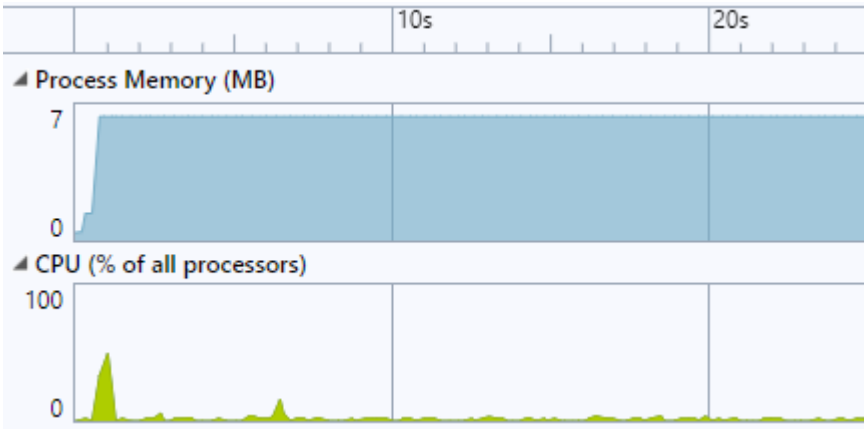


Obrázek 3.1: Ukázka mapování souboru do adresního prostoru procesu

Celé řešení se nachází ve třídě `File_mapping`. Tato třída mapuje soubor do paměti a rozděluje ho mezi výpočetními vlákny.

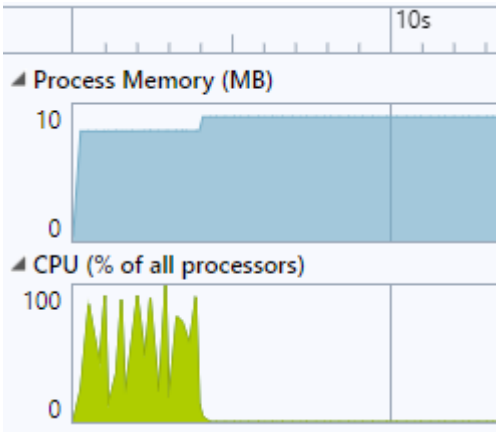
3.2 První řešení

V první verzi mapování a zpracování souboru jsem rozhodla dělit celý soubor na menší bloky ke zpracování. Program mapoval postupně každou menší část souboru a rozděloval ho mezi vlákny. Takové řešení mělo několik nevýhod. První bylo, že soubor ležící na disku typu HDD se načítal hodně pomalu (viz obrázek 3.2).



Obrázek 3.2: Ukázka využití CPU při prvním běhu. Zpracování souboru o velikosti 7GB, první běh s dobou trvání 10 min

Mnou mapované bloky se v prvním běhu programu nenacházely v paměti RAM, a proto vlákna zbytečně čekala na přesun stránky paměti z disku. Další běhy programu byly o hodně rychlejší (viz obrázek 3.3), nicméně podstatou programu není mnohonásobný běh.



Obrázek 3.3: Ukázka využití CPU při druhém běhu. Zpracování souboru o velikosti 7GB, každý další běh s dobou trvání cca 5 sec

Druhá nevýhoda byla v tom, že moje sekvenční řešení, které mapovalo celý soubor najednou, bylo vždycky rychlejší. Vzhledem k tomu, že paralelizace by měla urychlovat výpočet, nikoliv ho zpomalovat, jsem rozhodla řešit problém jiným způsobem.

3.3 První pokus optimalizaci

Z příkladu výše vyplývá, že doba trvání 10-ti minut pro první běh programu se vstupními daty 7 GB téměř nesplňuje zadání, kde je definované omezení doby běhu na 15 min. Proto jsem rozhodla dynamicky měnit násobek "allocation granularity" (viz sekci 3.5) pro vypočet mapovaného bloku. Pokud doba zpracování přesahuje dobu 5 sekund (stanoveno pomocí pokusného běhu), velikost bloku se změnší tak, aby vlákno nemuselo čekat přiliš dlouho na přesun stránky z disku na RAM. Místo cca 2 GB mapovacích bloků budou mapovány bloky velikosti odpovídající "allocation granularity". Takový způsob prokázal většího využití CPU. Navíc takto malé bloky by stačilo načítat pouze v prvním průchodu souborem. Druhý průchod by byl rychlejší, protože všechny stránky už by byly uloženy v paměti RAM.

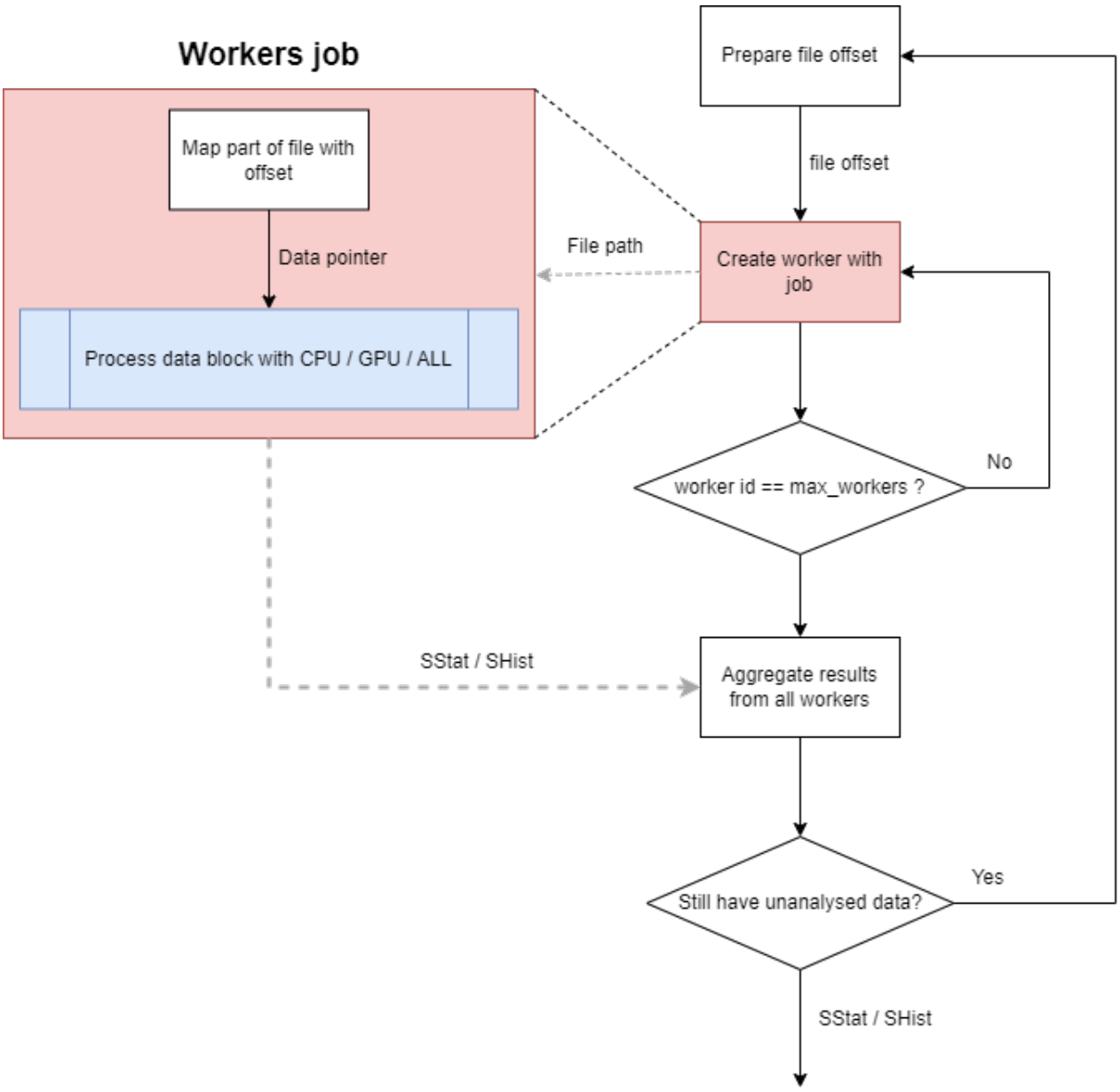
Bohužel tato optimalizace neprokázala vysokou spolehlivost výpočtu. Některá data se ztrácela a nedokázala jsem přijít na důvod.

3.4 Výsledné řešení - Režim s využitím CPU (smp)

V této sekci je popsané výsledné řešení mapování souboru a přidělování ho vláknům. Tímto způsobem se mi podařilo dosáhnout výrazného zrychlení programu v režimu SMP.

Program prochází vstupní soubor dvakrát. Algoritmus procházení souborem je ve dvou iteracích stejný (obrázek 3.4). Zpracování jednotlivých bloků souboru se provádí paralelně pomocí „worker“-ů. Liší se pouze funkce, se kterou každý worker pracuje. Při prvním běhu program sbírá statistiky dat, jako jsou celkový počet, suma, minimální a maximální hodnoty. Sbírané statistiky jsou potřebné pro sestavení histogramu a počítání parametrů rozdělení pomocí vzorců Maximum likelyhood estimation. Druhé procházení sestavuje frekvenční histogram, který se následně změní na histogram hustoty rozdělení a pomocí RSS metody bude zjištěno vstupní rozdělení. Pří každém procházení souborem se práce s jednotlivými bloky souboru rozděluje mezi vlákna, která jsou implementované pomocí std::async(std::launch::async). Každé vlákno samostatně namapuje blok souboru, se kterým bude pracovat.

Mapovaný soubor nezabírá žádnou paměť v adresním prostoru procesu, ale bylo nutné si alokovat místo na výstupní vektory pro výpočet statistik s použitím automatické vektorizace (viz sekce 3.4.1). Proto byla limitována velikost mapované části souboru pro jedno vlákno na 600 MB s ohledem na zadání semestrální práce. Tahle velikost se zarovná podle „allocation granularity“ systému.



Obrázek 3.4: Ukázka algoritmu mapování souboru pro všechny režimy

3.4.1 Sběr statistik

Jakmile si vlákno namapuje svůj blok dat do paměti, tak zavolá funkci `get_statistics_vectorized()`, která projde skrz celý blok pomocí cyklu **for** a získá potřebné lokální statistiky.

První varianta cyklu vypadala následujícím způsobem:

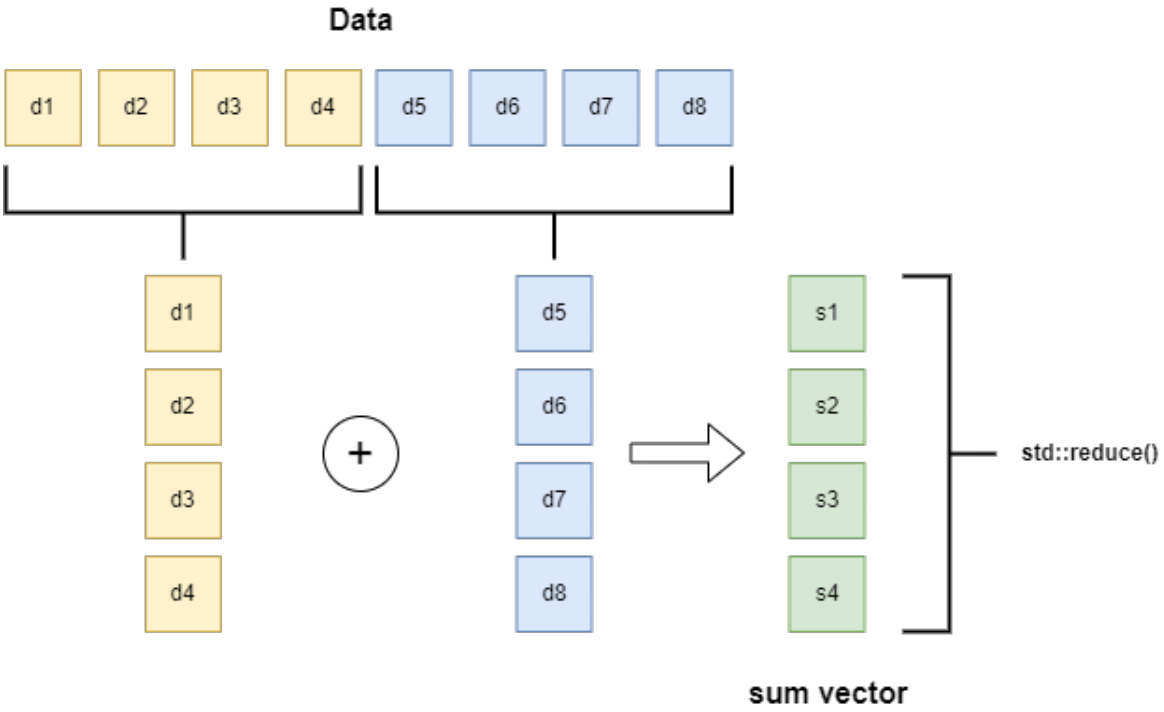
```
void get_statistics_vectorized(SDataStat& stat, long long
    data_count, double* data)
{
    long long n = 0;
    double sum = 0;
    double min = std::numeric_limits<double>::max();
    double max = std::numeric_limits<double>::min();

    for (int i = 0; i < data_count; i++)
    {
        n = n + 1;
        sum = sum + data[i];
        min = data[i] < min ? data[i] : min;
        max = data[i] > max ? data[i] : max;
    }

    stat.sum = sum;
    stat.n = n;
    stat.min = min;
    stat.max = max;
}
```

Nicméně kompilátor nedokázal rozpoznat redukci dat typu „double“ (kód 1105 [Mic21b]). Takový kód vyžaduje použití nastavení **/fp:fast**, což není součásti zadání. Proto bylo rozhodnuto rozdělit vstupní data na dvě části a provést operaci (suma, min a max) mezi dvěma poli (viz Obrázek 3.5). Řešení kompilátor rozpoznává a dokáže ho automatický zvektorizovat.

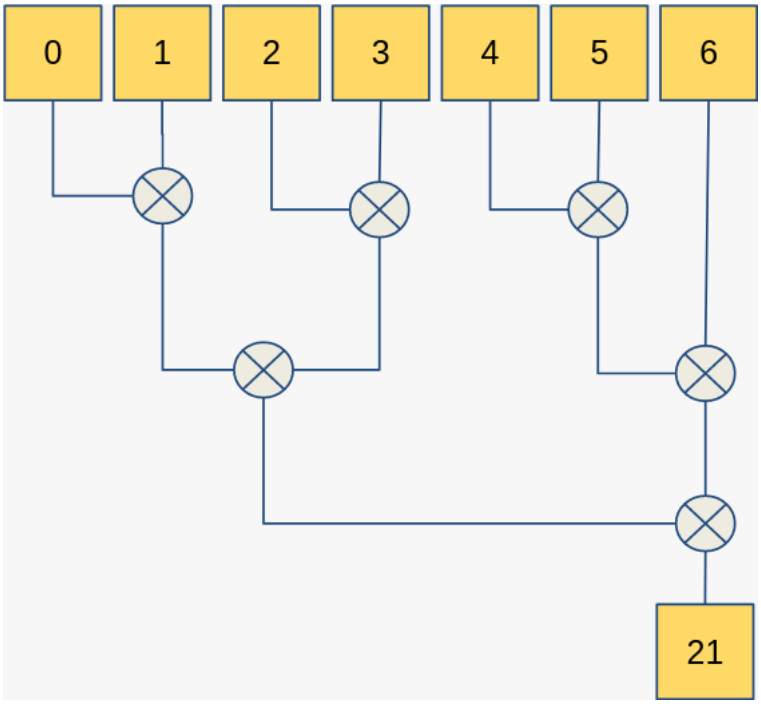
Problém ale nastává v tom, že výsledná suma (min a max) není skalární hodnota, ale vektor, který je nutný zredukovat.

Pro redukci vektoru sumy byl využit algoritmus standartní knihovny `std::reduce(std::execution::par_unseq, /**/)`, který nachází sumu vstupního vektoru s použitím vektorizace a paralelizace. Princip práce standartního algoritmu reduce je ukázán na Obrázku číslo 3.6. Redukce výsledných vektorů s maximálními a minimálními hodnoty se provádí pomocí standartního algoritmu `std::minmax_element( /**/)`.

Algoritmus `minmax_element` nepoužívá paralelní výpočet, ale ukázal se jako rychlejší řešení (oprotí `std::sort(std::execution::par_unseq, /**/)`) pro hledání maxima a minima.



Obrázek 3.5: Ukázka hledání sumy vektoru s auto-vektorizaci



Obrázek 3.6: Ukázka algoritmu std::reduce s použitím std::execution::par [@Ta18]

S použitím parametru kompilátoru `/Qvec-report:1` si lze ověřit funkčnost auto-vektorizaci.

```
1>--- Analyzing function: double __cdecl ppr::parallel::
   sum_vector_elements_vectorized(double * __ptr64,int)
1>D:\Study\ZCU\5.semestr\PPR\kiv-ppr\src\smp_utils.cpp(186) :
    info C5001: loop vectorized

1>--- Analyzing function: void __cdecl ppr::parallel::
   agregate_gpu_stat_vectorized(struct SDataStat & __ptr64,
   double * __ptr64,double * __ptr64,double * __ptr64,int)
1>D:\Study\ZCU\5.semestr\PPR\kiv-ppr\src\smp_utils.cpp(164) :
    info C5001: loop vectorized

1>--- Analyzing function: void __cdecl ppr::parallel::
   get_statistics_vectorized(struct SDataStat & __ptr64,
   __int64,double * __ptr64)
1>D:\Study\ZCU\5.semestr\PPR\kiv-ppr\src\smp_utils.cpp(137) :
    info C5001: loop vectorized

1>--- Analyzing function: void __cdecl ppr::executor::
   compute_propability_density_histogram(struct SHistogram &
   __ptr64,class std::vector<int,class std::allocator<int> >
   & __ptr64,class std::vector<double,class std::allocator<
   double> > & __ptr64,unsigned __int64)
1>D:\Study\ZCU\5.semestr\PPR\kiv-ppr\src\executor.cpp(51) :
   info C5001: loop vectorized
```

Upravený cyklus na sběr statistik vypadá následujícím způsobem (viz Obrázek 3.5):

```
void get_statistics_vectorized(SDataStat& stat, long long
    data_count, double* data)
{
    int half_count = data_count / 2;
    double* data_right = data;
    double* data_left = data + half_count;

    std::vector<double> sum(half_count);
    std::vector<double> min(half_count);
    std::vector<double> max(half_count);

    int count = data_count;

    for (int i = 0; i < half_count - 1; ++i)
    {
      sum[i] = data_right[i] + data_left[i];
      min[i] = data_right[i] < data_left[i] ? data_right[i] :
    data_left[i];
      max[i] = data_right[i] > data_left[i] ? data_right[i] :
    data_left[i];
    }

    stat.sum = std::reduce(std::execution::par_unseq, sum.
    cbegin(), sum.cend());
    stat.n = data_count;
    const auto [min_val, xxx1] = std::minmax_element(begin(
    min), end(min));
    const auto [xxx2, max_val] = std::minmax_element(begin(
    max), end(max));
    stat.min = *min_val;
    stat.max = *max_val;
}
```

3.4.2 Histogramy

Cyklus pro sestavení histogramu nelze zvektorizovat kvůli nesouvislému přístupu do paměti (chyba 1203 [Mic21b]). Po zjištění potřebné pozice do pole histogramu je nutné inkrementovat hodnotu na této pozici. Hodnota je náhodná a nedá se předpovědět jaká bude, protože je závislá na přečteném čísle. V tomto případě vektorizace nevede k urychlení počítaní, proto je v daném případě zbytečná.

Další cyklus, kde se podařilo výpočet zvektorizovat je nacházení histogramu hustoty (`compute_propability_density_histogram()`). Vzhledem

k tomu, že počet vstupních dat byl zredukován na menší pole histogramu, tento cyklus vždycky bude procházet pouze menší pole dat (cca 64 prvků u 16 GB vstupního souboru), proto daná vektorizace nevede k velkému zrychlení.

3.5 Výsledné řešení - Režim s využitím CPU a GPU (all)

Kódy pro grafické akcelerátory se nacházejí v samostatných souborech `statistics_kernel.cl` a `histogram_kernel.cl`.

Pro předání vstupních dat do grafického zařízení je nutné alokovat velké množství dat v adresovém prostoru procesu. Zároveň je nutné počítat s velkými výslednými poli, které zařízení vrací. Proto bylo rozhodnuto používat menší bloky souboru pro každý proces. Velikost mapovaného souboru byla zvolená na 500 MB. Limity byly zvolené metodou několika zkušebních běhů v respektu k omezením v zadání semestrální práce.

3.5.1 Sběr statistik

Pro počítání statistik na grafickém zařízení byl zvolen algoritmus **Parallel Sum Reduction**, který je popsaný na této stránce [DOU22], který byl doplněn o vyhledání maxima a minima.

Potom co vlákno namapuje vlastní blok dat pro zpracování, zavolá se funkce `run_statistics_on_GPU()`, která pro přidělené zařízení připraví buffery pro vstupní a výstupní data, dále připraví spouštěcí kód a provede výpočet. Velkou nevýhodou řešení je kopírování dat do bufferu, které se provádí pomocí CPU.

Algoritmus **Parallel Sum Reduction** vrací velká pole dat o velikosti počtů „work group“ (velikost dat / velikost jedné „work“ group-y). Pomocí CPU je pak nutné zredukovat výsledná pole, což se provádí pomocí zvektorizovaného cyklu ve funkci `agregate_gpu_stat_vectorized()`. Tento cyklus funguje na stejném principu jako byl popsán v sekci 3.4.1.

3.5.2 Histogramy

Sestavení histogramu nelze provést algoritmem redukce, proto se musí využit atomické instrukce, které budou navyšovat spočítanou frekvencí v globálním poli. Takové řešení zpomaluje proces, proto vypočet histogramu s grafickým zařízením trvá delší dobu. Kernel počítá také rozptyl rozdělení, jehož redukce se provádí pomocí vektorizované funkce `sum_vector_elements_vectorized`.

Sestavení histogramu hustoty se provádí stejným způsobem jako v režimu SMP.

3.6 Počítaní hodnot RSS

Posledním krokem výpočtu je vypočet RSS hodnot pro každé rozdělení na základě histogramu hustoty. Nejmenší výsledek RSS bude skutečné rozdělení vstupních dat. Tento krok je stejný u všech režimů

Hodnoty RSS pro všechny distribuce se počítá ve vláknech `std::async` (funkce `calculate_histogram_RSS_cpu` v souboru `smp_utils.cpp`). V souboru **rss.cpp** se nachází implementace počítaní RSS a také implementace funkci PDF pro každé rozdělení (viz sekci 2.4).

3.7 Watchdog

Watchdog je vlákno, které v nekonečné smyčce kontroluje správnost počítaných dat podle implementovaných pravidel. Jestli najde nesouvislost, ukončí program a upozorní na to uživatele. Celá implementace Watchdogu se nachází v souboru `watchdog.cpp`.

4 Uživatelská dokumentace

4.1 Ovládání

Pro spuštění programu musí být zadané povinné argumenty:

- Cesta k souboru s daty
- Režim programu
  - **smp**
  - **seq**
  - **all**
  - Názvy jednotlivých OpenCL zařízení

Mimo povinných vstupních argumentů programu jsem přidala následující nepovinné argumenty:

- **- o** argument musí být typu boolean. Nastavuje, zda se program spustí s provedenou optimalizací. Defaultní hodnota je nastavena na TRUE, tedy s běh s optimalizací.
- **-w** je argument celočíselného typu a ovlivňuje dobu uspání Watchdogu, zadáno v sekundách.

Příklad validního vstupu, který zapne program v režimu SMP bez optimalizace a watchdog vlákno bude kontrolovat běh programu každých 6 sekund:

```
1 "C:\gauss" smp -o 0 -w 5
```

4.2 Výstup

Po zpracování souboru program vždy vypíše následující informace:

- Vstupní parametry, se kterými program byl spuštěn
- Spočítané statistiky
- Výsledky RSS

- Čas běhu
- Výsledné rozdělení s parametry spočítané na základě nejnižší hodnoty RSS

Příklad výstupu programu v režimu SMP se vstupním souborem (7 GB) obsahující data s **normálním rozdělením**:

```
                        [Initial parameters]
-------------------------------------------------
> File:                           C:\delete\gauss
> Mode:                           smp
> Number of threads:              8
> Optimalization:                 TRUE
> Watchdog timer:                 2 sec


> Started ..

                        [Statistics]
-------------------------------------------------
> n:                              1000000000
> sum:                            -3572.2
> mean:                           -3.5722e-06
> variance:                       1.00005
> min:                            -6.10922
> max:                            5.87261
> isNegative:                     1
> isInteger:                      0


                        [Results]
-------------------------------------------------
> Gauss RSS:                      0.00079459
> Poisson RSS:                    -nan(ind)
> Exponential RSS:                0.588353
> Uniform RSS:                    0.477286



                        [Time]
-------------------------------------------------
> Statistics computing time:      3.02456 sec.
> Histogram computing time:       4.37617 sec.
> RSS computing time:             0.0003251 sec.
> TOTAL TIME:                     7.40135 sec.

> Input data have 'Gauss/Normal distribution' with mean
   =-3.5722e-06 and variance=1.00005
```

Příklad výstupu programu v režimu SMP se vstupním souborem (1 GB) obsahující data s **exponenciálním rozdělením**:

```
                        [Initial parameters]
-------------------------------------------------
> File:                           C:\delete\exp
> Mode:                           smp
> Number of threads:              8
> Optimalization:                 TRUE
> Watchdog timer:                 2 sec


> Started ..

                        [Statistics]
-------------------------------------------------
> n:                              137524224
> sum:                            1.37528e+08
> mean:                           1.00003
> variance:                       1.00015
> min:                            0
> max:                            20.9085
> isNegative:                     0
> isInteger:                      0


                        [Results]
-------------------------------------------------
> Gauss RSS:                      0.24959
> Poisson RSS:                    0.17973
> Exponential RSS:                0.121527
> Uniform RSS:                    0.558112


                        [Time]
-------------------------------------------------
> Statistics computing time:      4.54918 sec.
> Histogram computing time:       0.803888 sec.
> RSS computing time:             0.0002501 sec.
> TOTAL TIME:                     5.35362 sec.

> Input data have 'Exponential distribution' with lambda
   =0.99997
```

Příklad výstupu programu v režimu SMP se vstupním souborem (1 GB) obsahující data s **poissonovým rozdělením**:

```
                        [Initial parameters]
-------------------------------------------------
> File:                           C:\delete\poisson
> Mode:                           smp
> Number of threads:              8
> Optimalization:                 TRUE
> Watchdog timer:                 2 sec


> Started ..

                        [Statistics]
-------------------------------------------------
> n:                              141318656
> sum:                            1.41323e+08
> mean:                           1.00003
> variance:                       0.999858
> min:                            0
> max:                            10
> isNegative:                     0
> isInteger:                      1


                        [Results]
-------------------------------------------------
> Gauss RSS:                      0.0203551
> Poisson RSS:                    2.58775e-09
> Exponential RSS:                0.402114
> Uniform RSS:                    0.208506


                        [Time]
-------------------------------------------------
> Statistics computing time:      3.32517 sec.
> Histogram computing time:       0.688783 sec.
> RSS computing time:             0.0001357 sec.
> TOTAL TIME:                     4.01441 sec.

> Input data have 'Poisson distribution' with lambda=1.00003
```

Příklad výstupu programu v režimu SMP se vstupním souborem (1 GB) obsahující data s **rovnoměrným rozdělením**:

```
                        [Initial parameters]
-------------------------------------------------
> File:                           C:\delete\uniform
> Mode:                           smp
> Number of threads:              8
> Optimalization:                 TRUE
> Watchdog timer:                 2 sec


> Started ..

                        [Statistics]
-------------------------------------------------
> n:                              133684224
> sum:                            6.68395e+07
> mean:                           0.49998
> variance:                       0.0833297
> min:                            0
> max:                            1
> isNegative:                     0
> isInteger:                      0


                        [Results]
-------------------------------------------------
> Gauss RSS:                      3.82184
> Poisson RSS:                    7.97942
> Exponential RSS:                7.39103
> Uniform RSS:                    0.0357194


                        [Time]
-------------------------------------------------
> Statistics computing time:      4.0244 sec.
> Histogram computing time:       0.654257 sec.
> RSS computing time:             0.0001364 sec.
> TOTAL TIME:                     4.67911 sec.

> Input data have 'Uniform distribution' with a=0 and b=1
```

5 Analýza výsledků

Tabulka číslo 5.1 a graf číslo 5.1 ukazují zrychlení jednotlivých režimů programu oproti sekvenčnímu režimu. Podařilo se dosáhnout výrazného zrychlení vypočtu v režimem SMP s použitím auto-vektorizace.

Režimy s využitím GPU akcelerátorů jsou výrazně pomalejší, kvůli pomalému kopírování dat do zařízení a také kvůli atomickým operacím ve spouštěcím kernelu histogramu. Tohle lze pozorovat v tabulce číslo 5.3, kde je vidět, že výpočet histogramu tvoří větší část z celkového času běhu. Režim **ALL** je pomalejší kvůli čekání na nejpomalejší zařízení, což je "Intel(R) UHD Graphics 620".

Parallel - SMP (opt)	142%
"NVIDIA GeForce MX150"	12%
Parallel - ALL	-8%
"Intel(R) UHD Graphics 620"	-11%

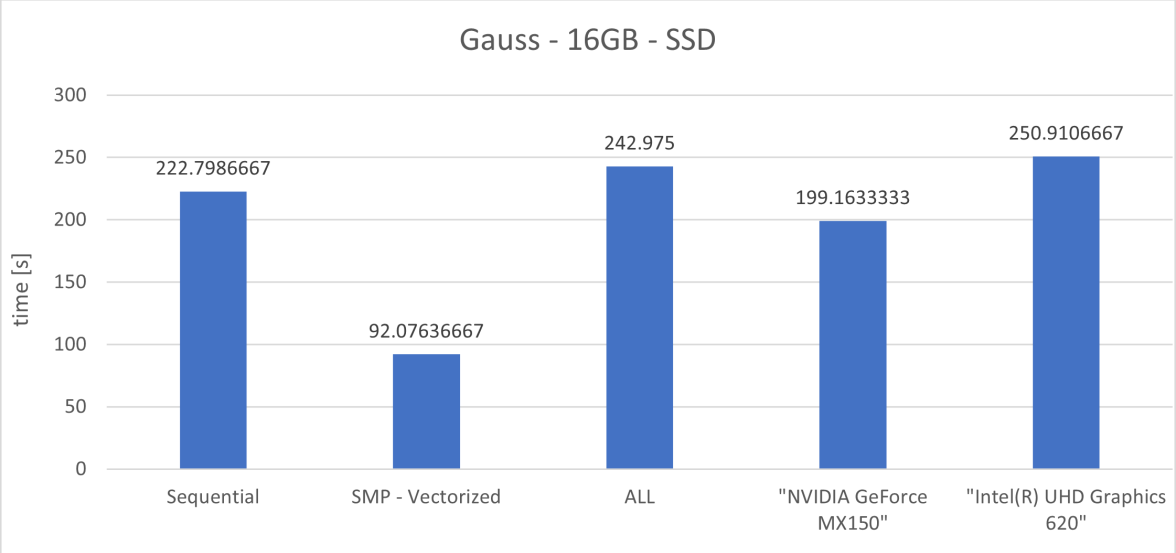
Tabulka 5.1: Porovnání rychostí jednotlivých režimů programu oproti sekvenčnímu režimu (velikost souboru 16 GB, SSD disk)

#	Total	Statistics	Histogram	RSS
1.	92.0155	54.023	36.9707	0.0043109
2.	92.2389	55.3738	36.352	0.0039542
3.	91.9747	54.2699	37.1852	0.0080428
Průměr	92.07636667	54.55556667	36.83596667	0.005435967

Tabulka 5.2: SMP-vektorizace. Ukázka času jednotlivých části programu. 16 GB, SSD disk)

#	Total	Statistics	Histogram	RSS
1.	229.045	60.7441	167.259	0.0002488
2.	248.838	82.5853	165.549	0.0003032
3.	251.042	73.167	176.814	0.000232
Průměr	242.975	72.165	169.874	0.000261333

Tabulka 5.3: Režim ALL. Ukázka času jednotlivých části programu. 16 GB, SSD disk)



Obrázek 5.1: Porovnání rychostí jednotlivých režimů programu (velikost souboru 7 GB, SSD disk)

6 Závěr

Zadání semestrální práce bylo splněno. Program vrací správný odhad pravděpodobnosti pro vstupní data na základě spočitané hodnoty RSS.

Program byl testován na operačním systému Windows 11 s procesorem Intel Core i7 a grafickou kartou NVIDIA GeForce MX150 a data se nacházely na SSD disku.

Nejvíce času jsem strávila vymýšlením algoritmu na detekci správného rozdělení. Další výzvou bylo porozumět, proč se nedaří využít plného vytížení CPU.

Tato práce mi vytvořila praktický přehled o tom, jak fungují vlákna a stránky paměti. Největší zpomalení programu dělají chybějící stránky v paměti RAM. Rychlost programu v režimu SMP také ovlivňuje, na jakém typu disku data leží.

Nepodařilo se mi dosáhnout rychlejšího vypočtu paralelizace s využitím OpenCL zařízení. Kopírování dat do zařízení zabírá moc času a paměti.

Nejrychlejším režimem programu se ukázal režim SMP s využitím automatické vektorizace. Všechna jednotlivá data a jejich porovnání, včetně porovnání rychlostí výpočtu, jsou dostupné v Excel tabulce ve složce s touto dokumentací.

Kód této práce lze najít v Git repositáři na této stránce https://bitbucket.org/pwnsauce8/kiv-ppr/src.

Literatura

[Bru18] BruceET. *Difference between Frequency and Density in a Histogram*. 2018. Available at `https://math.stackexchange.com/questions/2666834/what-is-the-difference-between-frequency-and-density-in-a-histogram`.

[DOU22] DOURNAC.ORG. *Parallel Sum Reduction*. 2022. Available at `https://dournac.org/info/gpu_sum_reduction`.

[Mic21a] Microsoft. *File Mapping*. 2021. Available at `https://learn.microsoft.com/en-us/windows/win32/memory/creating-a-view-within-a-file`.

[Mic21b] Microsoft. *Vectorizer and parallelizer messages*. 2021. Available at `https://learn.microsoft.com/en-us/cpp/error-messages/tool-errors/vectorizer-and-parallelizer-messages?view=msvc-170`.

[MT21a] PhD. Marco Taboga. *Exponential distribution - Maximum Likelihood Estimation*. 2021. Available at `https://www.statlect.com/fundamentals-of-statistics/exponential-distribution-maximum-likelihood`.

[MT21b] PhD. Marco Taboga. *Normal distribution - Maximum Likelihood Estimation*. 2021. Available at `https://www.statlect.com/fundamentals-of-statistics/normal-distribution-maximum-likelihood`.

[MT21c] PhD. Marco Taboga. *Poisson distribution - Maximum Likelihood Estimation*. 2021. Available at `https://www.statlect.com/fundamentals-of-statistics/Poisson-distribution-maximum-likelihood`.

[@Ta18] @TartanLlama. *std::accumulate vs. std::reduce*. 2018. Available at `https://blog.tartanllama.xyz/accumulate-vs-reduce/`.

[wik22a] wikipedia. *Kolmogorov–Smirnov$_t$est*. 2022. *Availableat*.

Wikipedia. *Residual sum of squares*. 2022. Available at `https://en.wikipedia.org/wiki/Residual_sum_of_squares`.