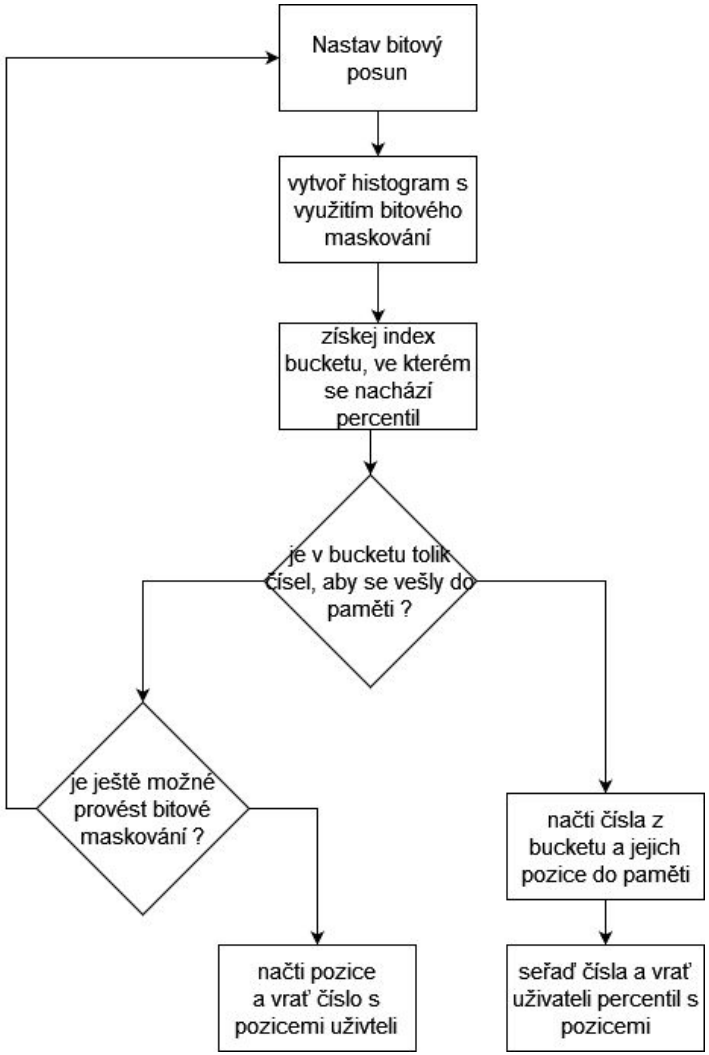
KIV/PPR

Semestrální práce

Přemysl Kouba
17.12.2021

Algoritmus

```
while (bucket.size > MAX_SIZE && is_maskable) {
        create_histogram();                          // by masking numbers
        bucket = find_bucket_index();                //where percentile belongs
}
return find_percentile_with_positions(index);        //percentile & positions in bucket
```


Nastav bitový posun
vytvoř histogram s využitím bitového maskování
získej index bucketu, ve kterém se nachází percentil
je v bucketu tolik čísel, aby se vešly do paměti ?
je ještě možné provést bitové maskování ?
načti pozice a vrať číslo s pozicemi uživteli
načti čísla z bucketu a jejich pozice do paměti
seřaď čísla a vrať uživateli percentil s pozicemi

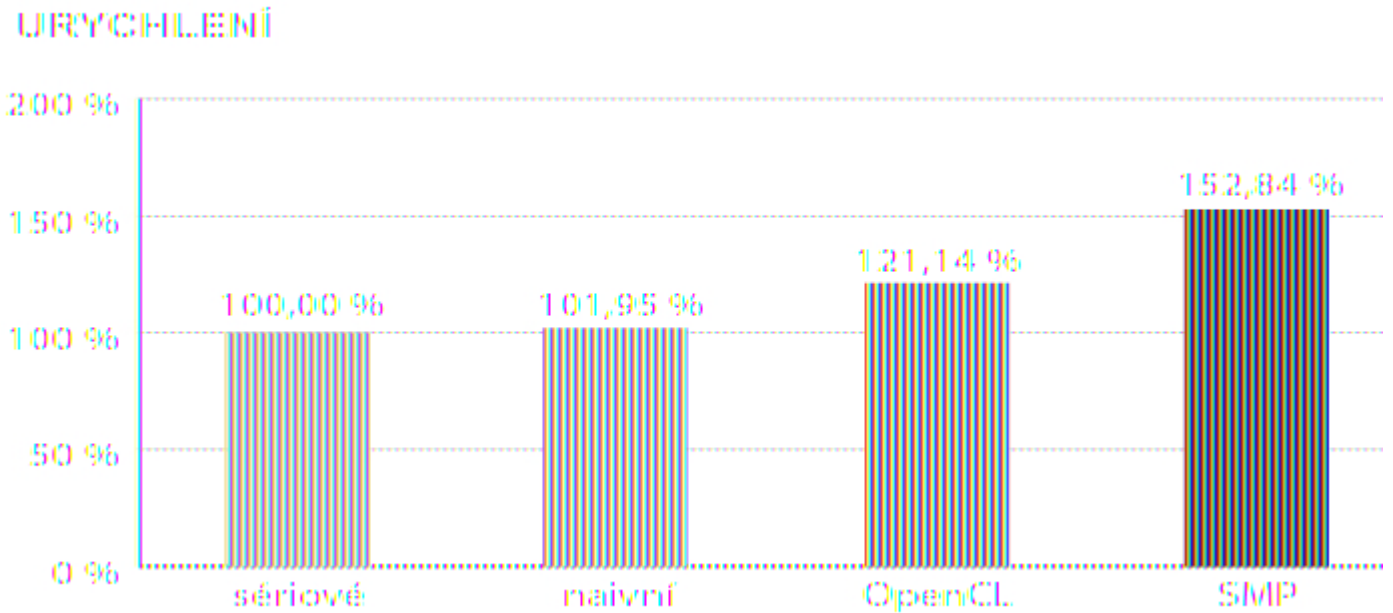
Implementace

C++

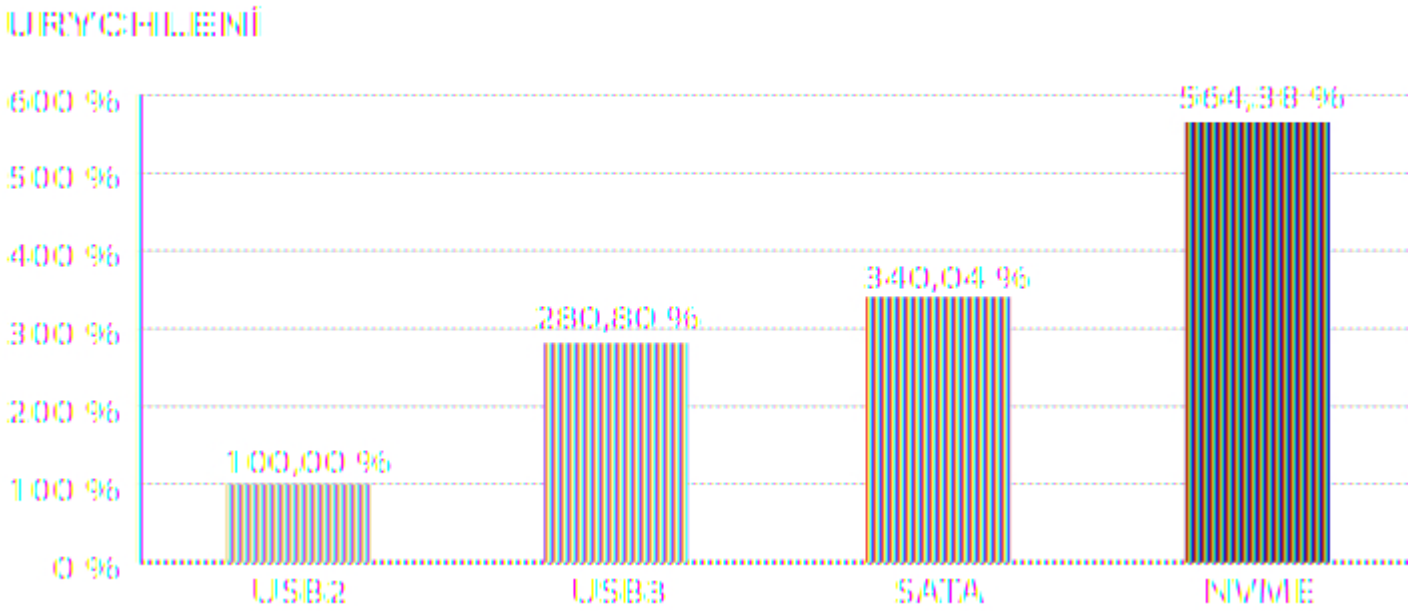
Intel TBB

OpenCL

Výsledky


ZRYCHLENÍ
200 %
150 %
100 %
50 %
0 %
100,00 %
101,965 %
121,14 %
152,84 %
sériově
naivní
OpenCL
SMP

Výsledky


UBYTCHLENÍ
6 000 %
5 000 %
4 000 %
3 000 %
2 000 %
1 000 %
0 %
100,000 %
280,880 %
340,004 %
5654,368 %
USB3.2
USB3.3
SATA
NVME

Výsledky

Typ zpracování	SUM (ms)	AVG (ms)	SPEEDUP%
naivní	2201016	22232,48485	101,95%
SMP	1468135	14829,64646	152,84%
sériové	2243848	22665,13131	100,00%
openCL	1852196	18709,05051	121,15%

SMP

- Paralelizace pomocí TBB (tbb::parallel_pipeline):
  - bitové maskování
  - hledání pozic pro konkrétní bucket
- Načítání po řádově menších částech než sériové řešení
- Následuje zpracování stejné jako u ostatních verzí
  - využívá paralelní std::sort pro nalezení výsledného percentilu

SMP-teoreticky

tbb::parallel_pipeline:

- bitové maskování:
  - sériové načtení souboru
  - paralelní maskování čísel do bucketů
  - složení histogramu na základě četností indexů (z předchozího kroku)
- hledání pozic (v konkrétním bucketu)
  - sériové načtení souboru
  - paralelní ukládání pozic čísel z bucketu do mapy (thread_safe)
  - sloučení čísel z předchozího kroku do vektoru

SMP-bitové maskování

```
tbb::parallel_pipeline(MAX_LIVE_TOKENS,
        tbb::make_filter<void, std::vector<double>>(
                tbb::filter_mode::serial_in_order, DataMiner(&parallelism_config, &file, &numbers_count_t)
        ) &
        tbb::make_filter<std::vector<double>, std::pair<uint64_t, std::vector<uint64_t>>>(
                tbb::filter_mode::parallel, DataMasker(&parallelism_config, watchdog)
        ) &
        tbb::make_filter<std::pair<uint64_t, std::vector<uint64_t>>, void> (
                tbb::filter_mode::serial_out_of_order, ChunkMerger(&h, watchdog, &mutex)
        )
        );
```

SMP-hledání pozic

```
tbb::parallel_pipeline(MAX_LIVE_TOKENS,
        tbb::make_filter<void, std::pair<size_t, std::vector<double>>>(
                tbb::filter_mode::serial_in_order, DataMinerWithPositions(&parallelism_config, &file)
        ) &
        tbb::make_filter<std::pair<size_t, std::vector<double>>, std::vector<double>>(
                tbb::filter_mode::parallel, DataMaskerPoistions(&positions, &parallelism_config, pr.bucket_index, watchdog)
        ) &
        tbb::make_filter<std::vector<double>, void>(
                tbb::filter_mode::serial_out_of_order, DataVectorMerger(&final_result, watchdog)
        )
        );
```

OpenCL

- Paralelizace pouze bitového maskování čísel na indexy bucketů
- Načítání po řádově menších částech než sériové řešení
- kernel funkce bucket_index()
  - vstupy:
    - buffer s daty
    - parametry (min, max, posun, maska)
  - výstupy:
    - vektor s indexy
      - UINT32_T_MAX pokud nepatří do bucketu
      - index bucketu
- Následuje zpracování stejné jako u ostatních verzí
  - využívá paralelní std::sort pro nalezení výsledného percentilu

OpenCL

```
__kernel void bucket_index(__global const double *data, __global uint *indexes, const uint shift, const int mask, const uint offset,
const double min, const double max) {
        int i = get_global_id(0);
        double value = data[i];
        uint index = max_int;
        if (is_valid(value) && is_in_range(value, min, max)) {
                long unsigned_value = as_long(value);
                if (value < 0) {
                        index = (~(unsigned_value >> shift)) & mask;                    //negative first
                } else {
                index = offset + ((unsigned_value >> shift) & mask); //negative first
                }
        }
        indexes[i] = index;
}
```

Shrnutí

- blokery:
  - problémy s C++
    - neměl jsem seminář (jen KIV/PC) => neumím ten jazyk
  - problémy s technologiemi
    - instalace nutného softwaru (Intel TBB, OpenCL headers, CMake, Visual Studio, …)
    - můj HW nepodporuje SYCL ve kterém jsem to chtěl dělat
- nedostatky řešení:
  - rozhodně není nejrychlejší řešení
  - OpenCL jen částečné
  - maskování čísel tak, aby indexy záporných čísel byly dříve

Shrnutí

- pozitiva:
  - sdílený kód napříč řešeními
    - snadnější údržba kódu a opravování chyb
    - větší přehlednost
  - program splňuje požadavky