Mikołaj Mrożewski
145331

Wyznaczanie liczb pierwszych CUDA

Data	Status Projektu	Uwagi
2.06.2022	Wybór tematu	
3.06.2022	Implementacja algorytmu sekwencyjnego	
5.05.2022	Implementacja algorytmu równoległego	
5.05.2022	Wykonanie pomiarów czasu	
6.06.2022	Analiza wyników	

Streszczenie

Liczby pierwsze od dawna są obiektem zainteresowania matematyków oraz informatyków. Algorytm sita Eratostenesa jest jednym z najbardziej znanych algorytmów do ich wyznaczania. Za pomocą CUDA można go zrównoleglić, i sprawdzić która wersja algorytmu będzie działać szybciej.

Opis problemu

Problem polega na zliczeniu liczb pierwszych w zakresie <2; n>, gdzie n jest podane przez użytkownika. Zadanie wykonałem w oparciu o algorytm sita Eratostenesa.

Algorytm sekwencyjny

Dane: zakres do 100 000, 500 000, 1 000 000, 5 000 000, 10 000 000, 50 000 000 oraz 100 000 000

Wyniki:

- dla 100 000: 9 592 liczby pierwsze
- dla 500 000: 41 538 liczb pierwszych
- dla 1 000 000: 78 498 liczb pierwszych
- dla 5 000 000: 348 513 liczb pierwszych
- dla 10 000 000: 664 579 liczb pierwszych
- dla 50 000 000: 3 001 134 liczby pierwsze
- dla 100 000 000: 5 761 455 liczb pierwszych

Metoda:

Algorytm zaczyna się od stworzenia tablicy wypełnionej jedynkami, poza indeksami 0 oraz 1 - te liczby nie mogą być liczbą pierwszą. Tablica ma długość równą zakresowi poszukiwania, który jest podany przez użytkownika. Każdy indeks tablicy jest kandydatem do bycia liczbą pierwszą. Iteruję po tablicy, sprawdzam czy na danym indeksie jest wartość 1. Jeśli tak jest, to każde pole w tablicy o indeksie będącym wielokrotnością tego indeksu zamieniane jest na wartość 0. Na koniec wywoływana jest funkcja zliczająca jedynki - liczby pierwsze - na liście.

Złożoność obliczeniowa: O($n * n * \sqrt{n} * log(n)$)
Złożoność pamięciowa: O(n)

Kody

Główna funkcja programu wielowątkowego

```
__global__ void generate_numbers(int range, int* numbers, int range_sqrt) {

    int beg = threadIdx.x * range / blockDim.x;
    int end = (threadIdx.x + 1) * range / blockDim.x - 1;

    //ones
    for (int i = beg; i <= end; i++) {
        if (i > 1)
            numbers[i] = 1;
    }

    __syncthreads();
    //sieve
    if (threadIdx.x == blockDim.x - 1) {
        for (int i = 2; i <= range_sqrt; i++) {
            if (numbers[i] == 1) {
                int temp_val = i * 2;
                while (temp_val <= range) {
                    numbers[temp_val] = 0;
                    temp_val += i;
                }
            }
        }
    }

    int counter = 0;
    __syncthreads();
    //counting
    for (int i = beg; i <= end; i++) {
        if (numbers[i] == 1) {
            counter += 1;
        }
    }

    __shared__ int counters[8];
    counters[threadIdx.x] = counter;
    __syncthreads();
    if (threadIdx.x == blockDim.x - 1) {
        counter = 0;
        for (int i = 0; i < blockDim.x; i++) {
            counter += counters[i];
        }
        printf("Primes in total: %d\n", counter);
    }

    __syncthreads();

}
```

Link do repozytorium:
https://github.com/mmrozewsk/Przetwarzanie_Rownolegle

Wynik programu:
Po uruchomieniu i wykonaniu obliczeń na wyjściu ukażą się ustawienia programu - zakres poszukiwań, liczba wątków - oraz czas obliczeń i ilość liczb pierwszych w danym zakresie. Poniżej przykładowy wynik dla zakresu poszukiwań do 100 000 000, oraz użycia dwóch wątków:

Primes in total: 5761455
Time: 112.749000, Range: 100000000, Threads:2

Testy programów i profilowanie aplikacji

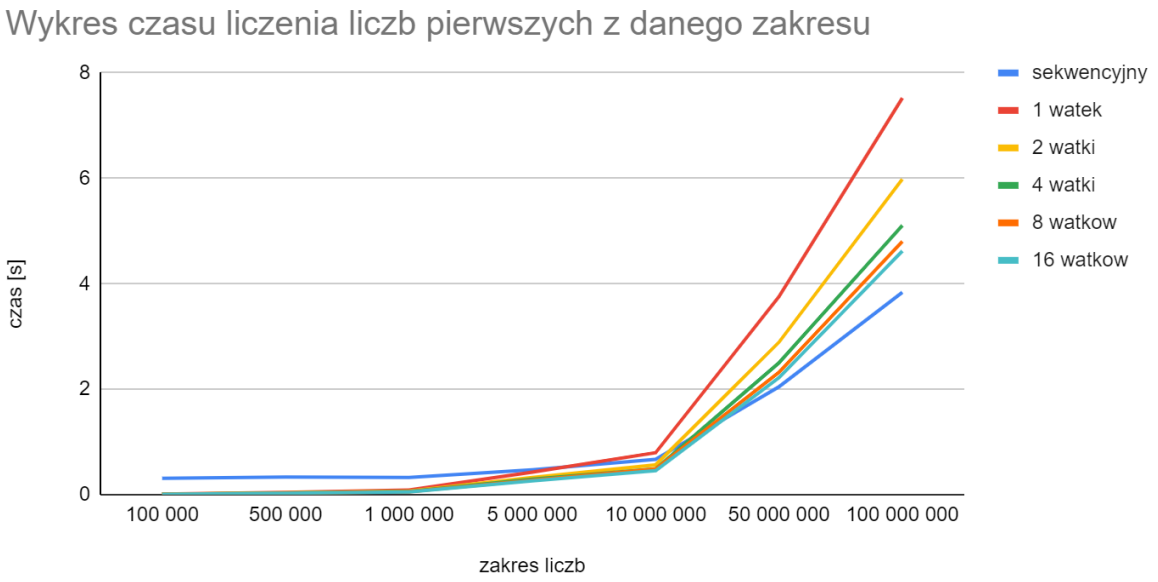
Podzespoły komputera:
- architektura 64 bitowa
- procesor Intel Core i7-7700HQ CPU @ 2.80GHz
- pamięć RAM 16GB

Sprawdzenie zgodności wyników wersji sekwencyjnej i równoległej algorytmów:
wyniki obu algorytmów są identyczne

Pomiary czasu

	sekwencyjny	1 wątek	2 watki	4 watki	8 wątków	16 wątków
100000	1,547	0,183	0,119	0,1	0,086	0,079
500000	1,552	0,801	0,596	0,485	0,401	0,376
1000000	1,518	1,54	1,096	0,891	0,776	0,742
5000000	1,614	7,565	5,444	4,364	3,814	3,552
10000000	2,011	15,216	10,956	8,782	7,685	7,115
50000000	3,534	77,806	55,898	44,936	39,43	36,688
100000000	5,104	155,543	112,749	90,837	79,919	74,409

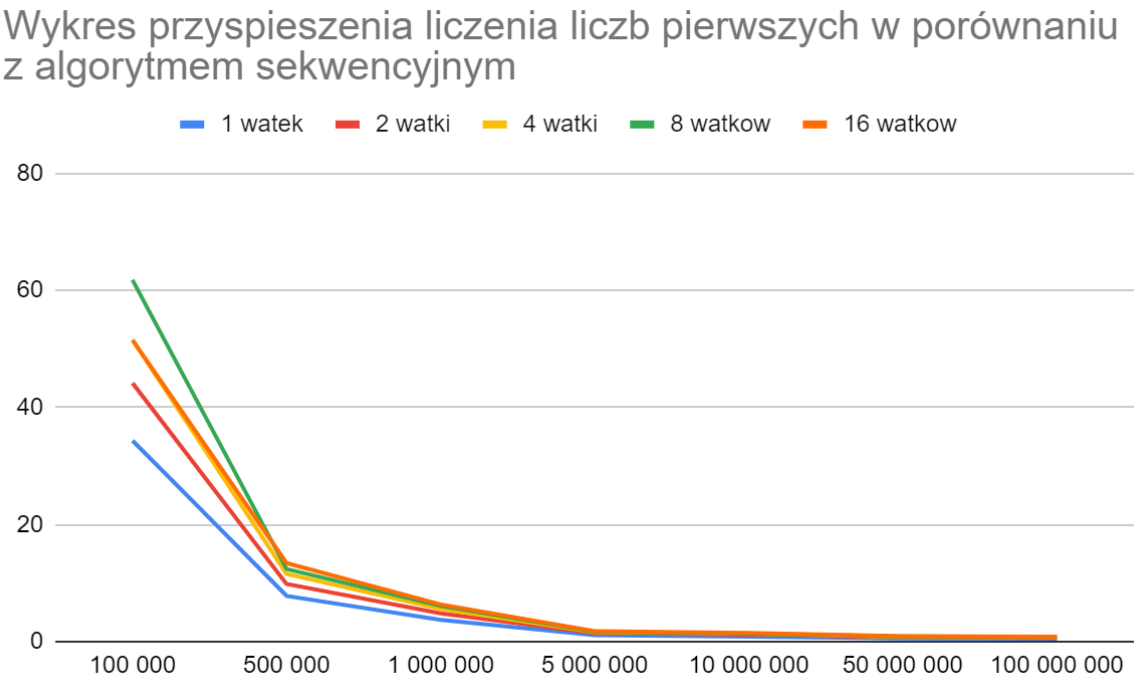
Wizualizacja pomiaru czasu



Analiza i wizualizacja narzutów czasowych i przyspieszenia obliczeń

Przyspieszenie wyliczane w stosunku do czasu wykonywania programu sekwencyjnego

Zakres	1 wątek	2 wątki	4 wątki	8 wątków	16 wątków
100 000	8,453551913	13	15,47	17,98837209	19,58227848
500 000	1,937578027	2,604026846	3,2	3,87032419	4,127659574
1 000 000	0,9857142857	1,385036496	1,703703704	1,956185567	2,045822102
5 000 000	0,2133509584	0,2964731815	0,3698441797	0,4231777661	0,4543918919
10 000 000	0,1321635121	0,1835523914	0,2289911182	0,2616785947	0,282642305
50 000 000	0,04542066165	0,06322229776	0,07864518426	0,08962718742	0,0963257741
100 000 000	0,03281407714	0,04526869418	0,05618855753	0,06386466297	0,06859385289



Analiza złożoności pamięciowej

Złożoność pamięciowa wynosi O(n) - korzystamy z n-elementowej listy kandydatów na liczbę pierwszą

Podsumowanie

Przy użyciu CUDY zliczanie liczb pierwszych wykorzystując wiele wątków nie zawsze przynosi korzyść czasową względem wykonania sekwencyjnego. Program wielowątkowy działa szybciej dla małego zakresu poszukiwań, a dla większych zakresów znacznie dłużej.

Źródła

https://pl.wikipedia.org/wiki/Prawo_Amdahla
https://pl.wikipedia.org/wiki/Sito_Eratostenesa