

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 liczb 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 jedynekami, 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:

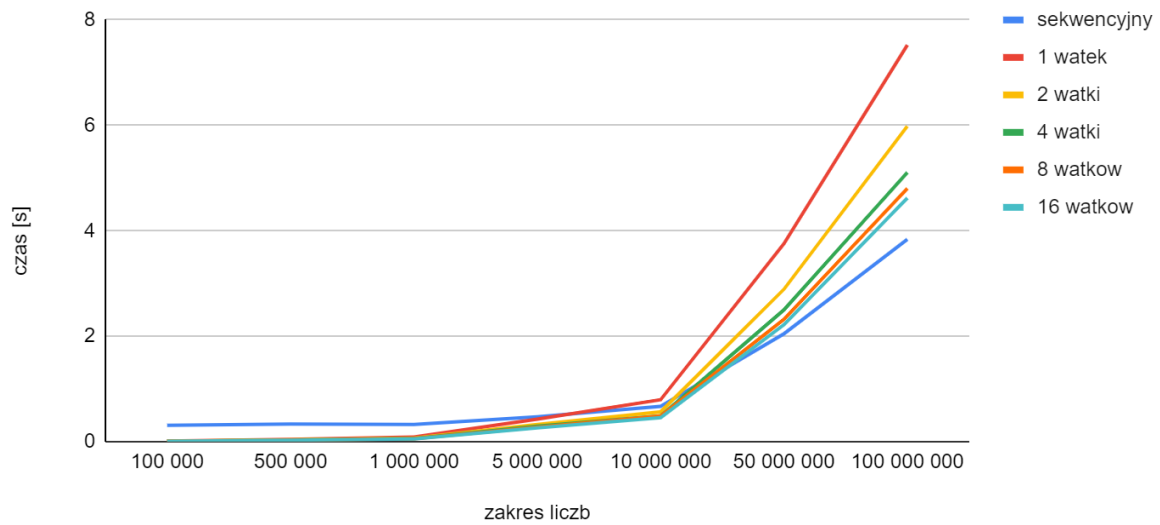
wyniki obu algorytmów są identyczne

Pomiary czasu

	sekwencyjny	1 watek	2 watki	4 watki	8 wątków	16 wątków
100000	0,309	0,009	0,007	0,006	0,005	0,006
500000	0,335	0,043	0,034	0,029	0,027	0,025
1000000	0,324	0,087	0,067	0,058	0,053	0,051
5000000	0,47	0,426	0,327	0,295	0,274	0,262
10000000	0,668	0,794	0,564	0,487	0,487	0,455
50000000	2,045	3,757	2,893	2,5	2,321	2,222
100000000	3,838	7,524	5,984	5,108	4,805	4,622

Wizualizacja pomiaru czasu

Wykres czasu liczenia liczb pierwszych z danego zakresu

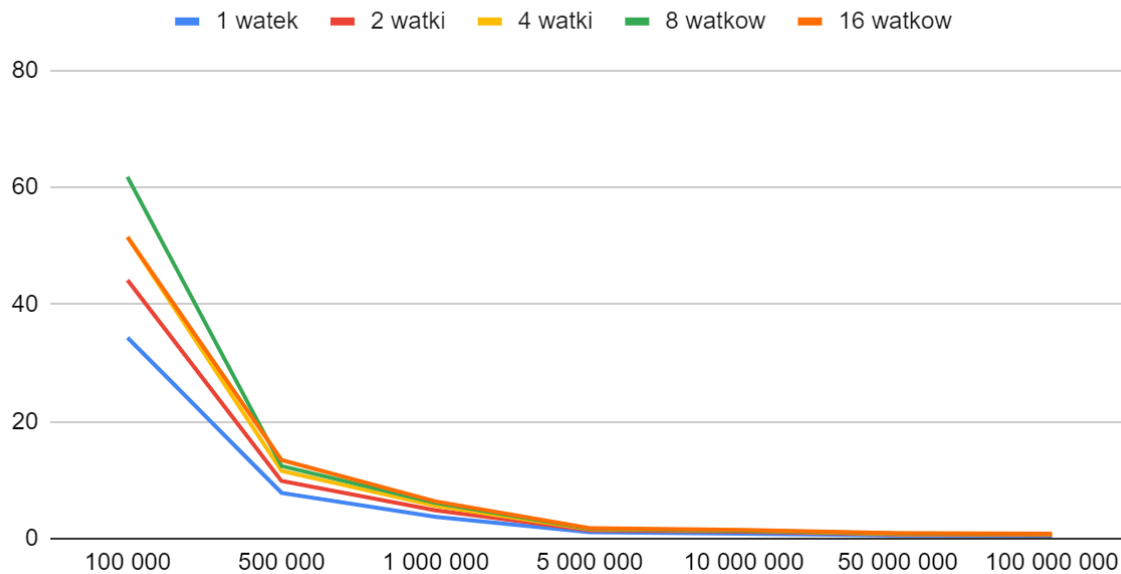


Analiza i wizualizacja narzutów czasowych i przyspieszenia obliczeń

Przyspieszenie wyliczane w stosunku do czasu wykonywania programu sekwencyjnego

Zakres	1 wątek	2 wątki	4 wątki	8 wątków	16 wątków
100 000	34,33333333	44,14285714	51,5	61,8	51,5
500 000	7,790697674	9,852941176	11,55172414	12,40740741	13,4
1 000 000	3,724137931	4,835820896	5,586206897	6,113207547	6,352941176
5 000 000	1,103286385	1,437308869	1,593220339	1,715328467	1,79389313
10 000 000	0,8413098237	1,184397163	1,371663244	1,371663244	1,468131868
50 000 000	0,5443172744	0,7068786727	0,818	0,8810857389	0,9203420342
100 000 000	0,5101010101	0,6413770053	0,7513703994	0,7987513007	0,8303764604

Wykres przyspieszenia liczenia liczb pierwszych w porównaniu z algorytmem sekwencyjnym



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