Gustaw Kustoń 145215

Wyznaczanie liczb pierwszych OpenMP

Data	Status projektu	Uwagi
2022-03-16	Wybór tematu	
2022-03-26	Implementacja algorytmu sekwencyjnego	
2022-03-27	Implementacja algorytmu równoległego	
2022-03-30	Porównanie wyników obu wersji algorytmu	
2022-04-01	Wykonanie pomiarów czasu	
2022-04-02	Analiza wyników cz. 1	
2022-04-04	Analiza wyników cz. 2	

Streszczenie

Efektywne szukanie liczb pierwszych to jedno z podstawowych problemów informatycznych. Znany algorytm sita Eratostenesa można efektywnie zrównoleglić w celu jego przyspieszenia.

Opis problemu

Problemem polega na znalezieniu i zliczeniu wszystkich liczb pierwszych w do danej liczby. Do tego zadania wykorzystano algorytm sita Eratostenesa. Algorytm ten opiera się na tablicy kandydatów, z której będziemy usuwać liczby, o których wiemy, że nie są pierwszymi. Dla każdej liczby zaczynając od dwójki usuwamy wszystkie liczby będące jej multiplikacją, które są większe lub równe kwadratowi tej liczby.

Opis algorytmu sekwencyjnego

- Algorytm badano dla liczb:
  100 000, 500 000, 1 000 000, 5 000 000, 10 000 000, 50 000 000, 100 000 000, 500 000 000, oraz 1 000 000 000
- Wyniki:
  100000 - 9592
  500000 - 41538
  1000000 - 78498

5000000 - 348513
10000000 - 664579
50000000 - 3001134
100000000 - 5761455

- Metoda: Wykorzystano standardową wersję algorytmu z tablicą, w której 1 oznacza potencjalną liczbę pierwszą, a 0 już odrzuconego kandydata. Pierwszy etap to inicjacja i wypełnienie tablicy jedynkami, drugi to odrzucenie liczb złożonych, a trzeci to zliczenie liczb pierwszych.
- Złożoność obliczeniowa: O(n (log n) (log log n))
  Złożoność pamięciowa: O(n)

Opis algorytmu równoległego

- Algorytm badano dla liczb:
  100 000, 500 000, 1 000 000, 5 000 000, 10 000 000, 50 000 000, 100 000 000, 500 000 000, oraz 1 000 000 000
- Wyniki:
  100000 - 9592
  500000 - 41538
  1000000 - 78498
  5000000 - 348513
  10000000 - 664579
  50000000 - 3001134
  100000000 - 5761455
- Metoda: Algorytm w wersji równoległej zyskuje na efektywności w dwóch miejscach. W faktycznym algorytmie wykorzystano dyrektywę #pragma omp parallel for schedule(dynamic), która umożliwia jednoczesne manipulowanie tą sama tablicą na róœnych indeksach.
  Podczas liczenia liczb pierwszych z tablicy wykorzystano dyrektywę #pragma omp parallel for reduction(+:res), co umożliwia poprawne i szybsze zliczanie przez wiele procesów do jednej zmiennej.
- Złożoność obliczeniowa: O(n (log n) (log log n))
  Złożoność pamięciowa: O(n)

Kody

- Fragmenty kodów:
  Główna część algorytmu z wykorzystaniem równoległości:

```
int isPrime(int num)
{
    int i = 2;
    while (i * i <= num)
    {
        if (num % i == 0)
        {
            return 0;
        }
        i++;
    }
    return 1;
}

int eratosthenes(int n, int num_threads)
{

    omp_set_num_threads(num_threads);

    int *primes = (int *)malloc(n * sizeof(int));

#pragma omp parallel for schedule(dynamic)
    for (int i = 0; i < n; i++)
    {
        primes[i] = isPrime(i);
    }

    int res = 0;
#pragma omp parallel for reduction(+ \
                                   : res)
    for (int i = 2; i < n; i++)
    {
        res += primes[i];
    }
    return res;
}
```

- Link do repozytorium projektu:
  https://github.com/gustkust/Primes-OpenMP
- Kompilacja: opcje kompilacji:
  gcc -o seq -fopenmp primes_seq.cpp -lstdc++ -lm
  gcc -o par -fopenmp primes_par.cpp -lstdc++ -lm
- Rozmiar kodu:

Testy programów i profilowanie aplikacji

- Architektura wykorzystanego komputera:
  Komputer działa w architekturze 64 bitowej, procesor to Intel(R) Core(TM) i5-6200U CPU @ 2.30GHz, a ilość ramu to 8 GB.
- Sprawdzanie zgodności wyników wersji sekwencyjnej i równoległej algorytmów:
  Algorytm w wersji sekwencyjnej i równoległej cechują się takimi samymi wynikami.

Pomiary czasu

Wyniki pomiarów czasu:

Liczba	Sekwencyjna	1 proces	2 procesy	4 procesy
100000	0.029586	0.066303	0.070165	0.112253
500000	0.138629	0.143504	0.112243	0.068156
1000000	0.279603	0.366456	0.163459	0.148305
5000000	2.586712	3.20309	1.341477	1.292793
10000000	7.771968	6.802665	4.182619	3.424452
50000000	70.856859	68.123688	35.398444	33.257224
100000000	183.69451	173.688277	93.354921	88.45684

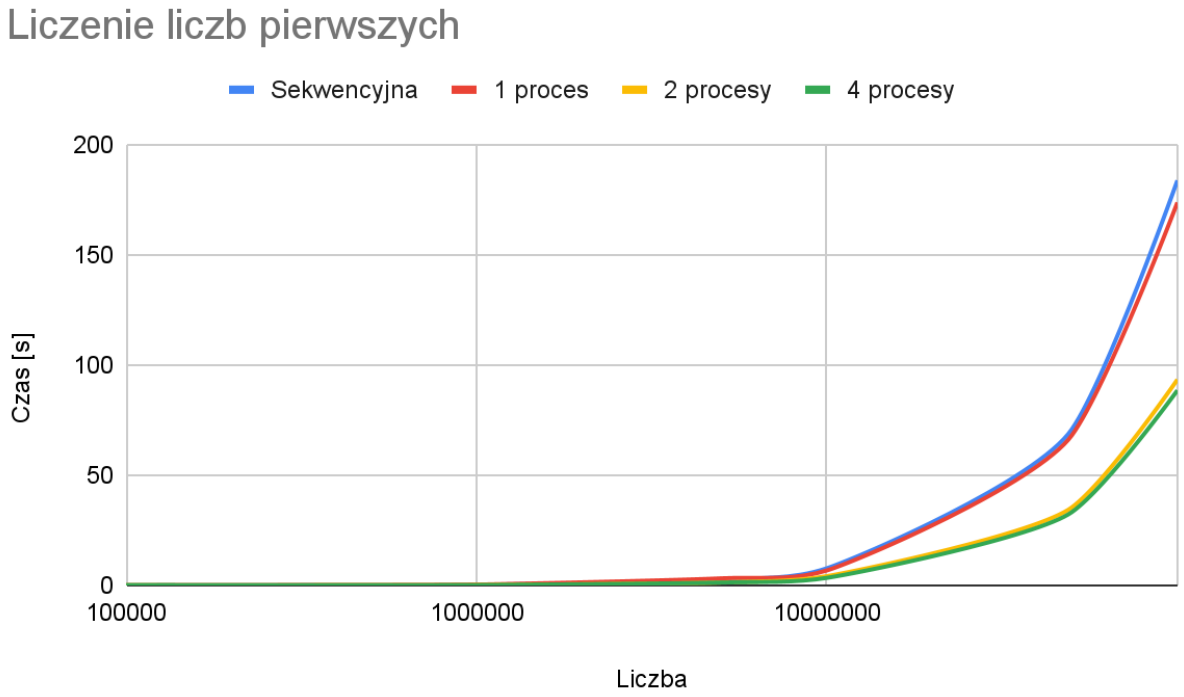
Analiza narzutów czasowych i przyspieszenia obliczeń

Przyspieszenie w stosunku do liczby procesów / wątków (prawo Amdahla i Gustafsona z wiki)

Wartości przyspieszenia:

Liczba	1 proces	2 procesy	4 procesy
100000	0.4462241528	0.4216632224	0.263565339
500000	0.9660288215	1.235079248	2.03399554
1000000	0.76299201	1.710539034	1.885324163
5000000	0.8075676924	1.92825669	2.000870982
10000000	1.142488716	1.85815825	2.269550865
50000000	1.040120714	2.001694171	2.130570459
100000000	1.057610296	1.967700342	2.076656932

Pomiary czasów na wykresie:



Wykorzystanie większej ilości procesów charakteryzuje się większą efektywnością.

Analiza złożoności pamięciowej

Program ma złożoność pamięciową O(n), ponieważ korzystamy z jednej tablicy przechowującej n wartości.

Podsumowanie

Poprzez zrównoleglenie operacji można dużo efektywniej znajdować i liczyć liczby pierwsze. Nim więcej procesorów tym lepsze wyniki, ale nie jest to zależność liniowa, ponieważ wszystkie procesy działają na jednej z tablicy, z tym że z jej różnych indeksach w różnych momentach przetwarzania. Oznacza to, że procesy nie mogą podzielić tablicy na równe części i wykonywać obliczenia lokalne, tylko ich działanie musi być dynamicznie skoordynowane

Źródła

https://en.wikipedia.org/wiki/Sieve_of_Eratosthenes

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

https://pl.wikipedia.org/wiki/Prawo_Gustafsona

http://jakascorner.com/blog/2016/06/omp-for-scheduling.html

https://docs.microsoft.com/en-us/cpp/parallel/openmp/reference/openmp-directives?view=msvc-170