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

Tutaj streszczenie, że generalnie liczby pierwsze się szuka

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
500000000 - 26355867
1000000000 - 50,847,534

- 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
  500000000 - 26355867
  1000000000 - 50,847,534
- Metoda: Algorytm w wersji równoległej zyskuje na efektywności w trzech miejscach.
  Podczas inicjacji tablicy wykorzystano dyrektywę #pragma omp parallel for do szybszego wypełnienia tablicy jedynkami.
  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 rangeSQRT = (int) sqrt((double) range);
  #pragma omp parallel for schedule(dynamic)
  for (int i = 2; i < rangeSQRT; i += 1)
          if (primes[i] == 1)
                  for (int j = i * i; j < range; j += i)
                          primes[j] = 0;
  ```
- 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	0.009899	0.008482	0.007098	0.061544
500000	0.026738	0.020084	0.014705	0.026542
1000000	0.039912	0.040062	0.020936	0.039237
5000000	0.16739	0.175399	0.128582	0.11481
10000000	0.294634	0.286646	0.217731	0.183054
50000000	1.762706	1.588963	1.082993	0.96271
100000000	3.194553	3.108211	2.262133	2.073297
500000000	17.750543	16.624099	11.933782	11.115678
1000000000	35.454692	34.893295	25.163177	23.780049

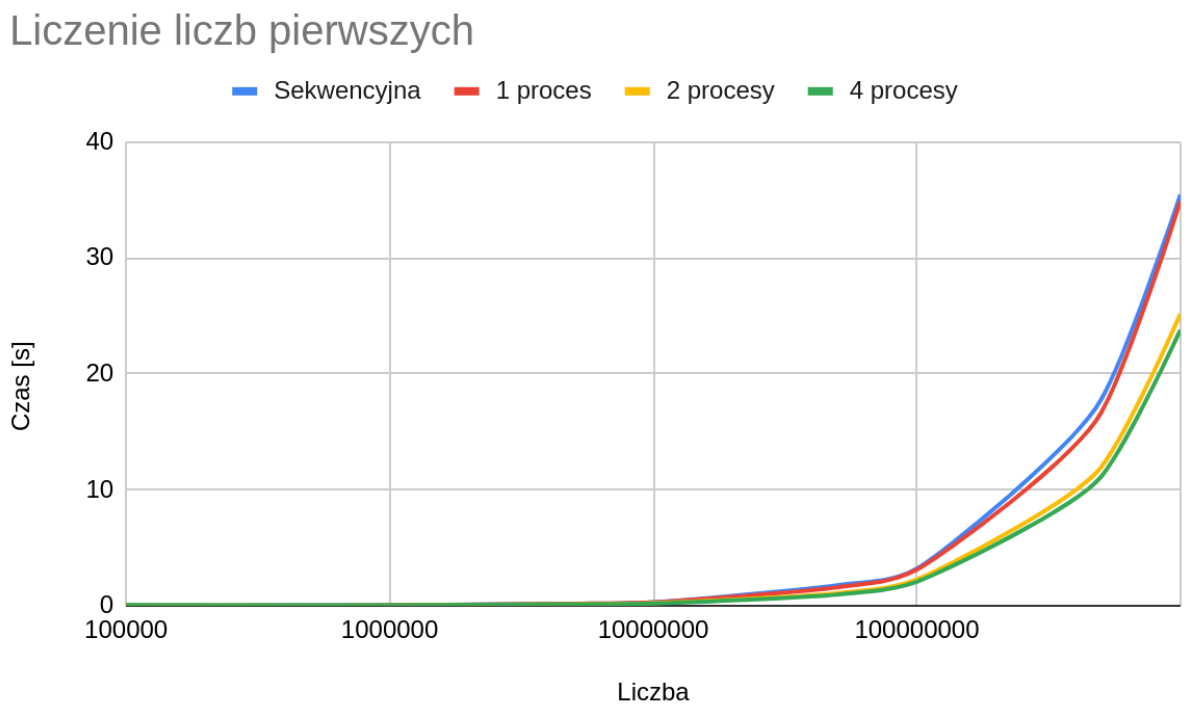
Analiza narzutów czasowych i przyspieszenia obliczeń

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

Wartości przyspieszenia:

Liczba	1 proces	2 procesy	4 procesy
100000	1.167059656	1.394618202	0.160844274
500000	1.331308504	1.818293098	1.007384523
1000000	0.9962558035	1.906381353	1.01720315
5000000	0.9543383942	1.301815184	1.457974044
10000000	1.027867125	1.353201887	1.60954691
50000000	1.109343641	1.627624555	1.83098337
100000000	1.02777868	1.412186198	1.540808191
500000000	1.067759702	1.487419747	1.596892515
1000000000	1.016088965	1.408991082	1.490942765

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