# Programowanie Równoległe - Liczby Pierwsze

Wersja pierwsza


### Autorzy

**Grupa dziekańska:** 4
**Grupa labolatoryjna:** 7
**Termin zajęć:** czwartek, 16:50

Tymoteusz Jagła 151811 - tymoteusz.jagla@student.put.poznan.pl
Kaper Magnuszewski 151746 - kacper.magnuszewski@student.put.poznan.pl


### Sprawozdanie

*Wymagany termin oddania sprawozdania* - 10.05.2024
*Rzeczywisty termin oddania sprawozdania* - 10.05.2024

## Opis zadania

Projekt polegał na zbadaniu efektywności przetwarzania równoległego w komputerze z procesorem wielordzeniowym. Badanym zadaniem było znajdowanie liczb pierwszych w określonym zakresie. Podano różne warianty algorytmów - wyznaczanie liczb pierwszych metodą dzielenia oraz przy użyciu sita Erastothenes'a, w tym podejście sekwencyjne oraz równoległe (domenowe i funkcyjne).

## Wykorzystany system obliczeniowy

### Procesor

- Model: 13th Gen Intel® Core(TM) i5-13600KF
- Liczba procesorów fizycznych: 14
- Liczba procesorów logicznych: 20
- Oznaczenie typu procesora: KF
- Wielkość pamięci podręcznej: 24 MB
- Organizacja pamięci podręcznej: Intel® Smart Cache

### System Operacyjny

- Nazwa systemu operacyjnego: Linux Pop!-OS 6.8.0
- Oprogramowanie wykorzystane do przygotowania kodu wynikowego: Visual Studio Code
- Oprogramowanie wykorzystane do przeprowadzenia testów: Intel VTune Profiler

## Wersje programów

### Zdefiniowane stałych używanych w kodzie

W osobnym pliku nagłówkowym zostały zdefiniowane stałe takie jak: - Dolna granica poszukiwania liczb pierwszych `M_VAL` - Górna granica poszukiwania liczb pierwszych `N_VAL` - Liczba procesorów użyta do wykonania zadania równolegle

### Liczby pierwsze wyznaczane sekwencyjnie przez dzielenie w zakresie <m, n> (k1)

Poniższy kod to podejście sekwencyjne. Mierzony jest czas pracy procesora za pomocą zmiennych `spstart` i `spstop` oraz rzeczywisty czas pracy programu za pomocą `sswtime` i `sewtime`. Tablica `primeArray` przechowuje zmienne typu `bool` - pierwiastki liczby `n`, które są liczbami pierwszymi. Program metodą dzielenia wyznacza tablicę `primeArray`, po czym korzystając z wartości do niej wpisanych sprawdza wszystkie liczby z zakresu podanego w pliku nagłówkowym. Jeżeli dana liczba nie jest podzielna przez żaden z podzielników `n`, oznacza to, że jest to liczba pierwsza. W takim wypadku jest ona zapisywana do tablicy wynikowej `result`.

### Liczby pierwsze wyznaczane równolegle przez dzielenie w zakresie <m,n> (k2)

Poniższy blok to równoległa implementacja kodu z poprzedniego zadania. W tym celu użyta została biblioteka `OpenMP`. W określonym obszarze równoległym, każdemu z wątków zostaje przydzielona wartość iteratora pętli `i`, który przyjmuje wartości w przedziale <m, n>. W tej wersji programu możemy ustawić różne wartości klauzuli `schedule`. Przy wartości `static` wątkom zostaną przydzielone różne liczby z zakresu <m, n>, co oznacza, że niektóre wątki zbadają mniej liczb niż inne, jeżeli badane liczby będą wyjątkowo duże. Ten przydział może okazać się niesprawiedliwy i wpłynąć na czas wykonywania programu.
Jeżeli wątkom przydzielone zostałyby kolejne wartości `i` mógłby również wystąpić false-sharing. Oznacza to, że wątki mogłyby nadpisywać tą samą linię pamięci, co znacząco spowolniłoby program. W tym wypadku false-sharing pojawiłby się poprzez nadpisywanie części tablicy `result` znajdujących się w tych samych liniach pamięci.

### Sito sekwencyjne bez lokalności dostępu do danych (k3)

Poniższy kod to sekwencyjna implementacja algorytmu sita Erastothenesa. Pierwsza pętla programu ponownie wyznacza liczby pierwsze w zakresie <2, n>. Druga pętla identyfikuje liczby pierwsze w zakresie <m, n> i oznacza jako liczby

złożone ich wielokrotności. W ten sposób w tablicy wynikowej otrzymujemy jedynie liczby pierwsze.

### Sito sekwencyjne z potencjalną lokalnością dostępu do danych (k3a)

### Sito równoległe funkcyjne bez lokalności dostępu do danych (k4)

Poniższy kod to równoległa implementacja poprzedniego zadania znajdowania liczb pierwszych przy użyciu algorytmu sita Erastothenes'a. W podanym algorytmie występuje brak lokalności dostępu do danych, co spowodowane jest możliwością odczytywania przez wątki komórek tablicy `result`, które dzielą tą samą linię cache. Działanie programu może zostać znacząco spowolnione.

### Sito równoległe funkcyjne bez lokalności dostępu do danych (k4a)

Jest to implementacja kodu z zadania poprzedniego, w której dodatkowo sprawdzana jest wartość tablicy `result` w komórce, do której program chciałby wpisać wartość `false`. Oznacza to, że możemy uniknąć wielu nadmiernych unieważnień pamięci poprzez ograniczenie ilości prób niepotrzebnych zmian wartości tablicy wynikowej.

### Sito równoległe domenowe z potencjalną lokalnością dostępu do danych (k5)

Poniższy kod również przedstawia algorytm sita Erastothenesa, jednak dodatkowo stosuje taktykę podziału obszaru roboczego na bloki. Dzięki takiemu rozwiązaniu możemy uniknąć false sharingu, a co za tym idzie znacznie przyspieszyć wykonywanie się programu. Najlepszy efekt osiągniemy, jeżeli wielkość bloku dostosujemy do długości linii pamięci. Dzięki temu unikniemy niepotrzebnych odczytów w linii pamięci przez różne wątki, a co za tym idzie jej unieważniania.