|  |  |  |
| --- | --- | --- |
| Łukasz Oprych  Gr.5 ITE | Programowanie Równoległe | Laboratorium 5,  Dekompozycja |

Cel ćwiczenia:

Zapoznanie się z mechanizmami dekompozycji, blokowej oraz cyklicznej w programach równoległych.

Przebieg ćwiczenia:

Po skonfigurowaniu struktury katalogowej i plików źródłowych zgodnie z poleceniami prowadzącego, skompilowano plik pthreads\_całka.c i dokonano pomiarów czasów obliczeń dla danej ilości wątków i rozmiaru danych w trzech wersjach: sekwencyjnej, ż użyciem mutexów, z użyciem tablicy.

Przykładowy wynik z konsoli:

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

Czasy dla wersji z tablicą o rozmiarze 1000 i 100000000 i ilości wątków 1,2,4, wersja zoptymalizowana -O3 :

Obraz zawierający tekst, zrzut ekranu, Czcionka, numer

Opis wygenerowany automatycznie

Czasy dla wersji z tablicą o rozmiarze 1000 i 100000000 i ilości wątków 1,2,4, wersja niezoptymalizowana -g ddebug :

Obraz zawierający tekst, zrzut ekranu, numer, Czcionka

Opis wygenerowany automatycznie

Kolejnym krokiem było odkomentowanie informacji na temat liczby wątków używanych w programie w pliku obliczanie\_calki.c ukrytych pod zmienną l\_w.

Fragment kodu wraz z sekwencyjnym obliczaniem całki prezentuje się w następujący sposób:

Obraz zawierający tekst, zrzut ekranu

Opis wygenerowany automatycznie

Następnym krokiem było zedytowanie pliku dekompozycja pętli.c.

Tworzenie mutexa oraz przyrównanie zmiennych globalnych:

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

Utworzenie struktur danych do obsługi wielowątkowości, tworzenie tablicy globalnej do wersji bez sekcji krytycznej, tworzenie wątków w pętli, zdefiniowanie joina w celu oczekiwania na zakończenie pracy wątków, oraz sumowanie tablicy calka\_global, w której każdy wątek ma osobny element i jest przekazywany do wyniku przez wątek główny.

Obraz zawierający tekst, zrzut ekranu, Czcionka, oprogramowanie

Opis wygenerowany automatycznie

Następnie w funkcji calka\_fragment\_petli zdefiniowanie dekompozycji blokowej. Dekompozycja blokowa to technika podziału danych na mniejsze bloki w celu łatwiejszego zarządzania i implementacji. Jeżeli rozmiar danych jest podzielny przez liczbę wątków, każdy wątek dostaje blok o tym samym rozmiarze danych.

Obraz zawierający tekst, Czcionka, zrzut ekranu, design

Opis wygenerowany automatycznie

Zmienna J odpowiada za obliczenie ilości danych przypisanych do każdego wątku bazując na ilości zadań oraz ilości wątków, funkcja ceil zaokrągla wynik dzielenia w góre, w celu optymalnego rozkładu bloków danych między wątkami, aby ostatni wątek nie został obciążony nadmiernie. Jako start dekompozycji bierzemy nr wątku \* ilość danych, końcem danego bloku jest nr wątku + 1 \* ilość danych, każdy krok w dekompozycji blokowej my\_stride wynosi 1. Mamy też tu warunek dla ostatniego wątku w przypadku niepodzielnego rozmiaru danych względem wątków.

Zdefiniowanie dekompozycji cyklicznej, w przypadku tej dekompozycji dane między wątkami są rozmieszczane co n-ty wątek, czyli np. jeżeli mamy n wątków, to co n-ty element jest przypisywany do danego wątku (1 element do pierwszego wątku, 2 element do drugiego wątku, 3 element do trzeciego wątku, 4 element do pierwszego wątku, 5 element do 5 wątku itd.)

Obraz zawierający tekst, Czcionka, zrzut ekranu, numer

Opis wygenerowany automatycznie

Zmienna my\_start określa pierwszy element wątku jako numer wątku, my\_id, końcem jest N, czyli rozmiar danych, kolejnym krokiem my\_stride jest liczba wątków.

Dokończenie funkcji calka\_fragment\_petli

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

Dokonanie pomiarów oraz testowanie dokładności rozwiązania dla różnych wysokości trapezu (dx) na podstawie dekompozycji blokowej.

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatyczniePrzykładowy wynik z konsoli:

Średnie wyniki pomiaru dla dx = 0.1 dla kolejno 1,2,4 wątków

Obraz zawierający tekst, paragon, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

Średnie wyniki pomiaru dla dx = 0.01 dla kolejno 1,2,4 wątków

Obraz zawierający tekst, paragon, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

Średnie wyniki pomiaru dla dx = 0.0001 dla kolejno 1,2,4 wątków

Obraz zawierający tekst, paragon, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

Średnie wyniki pomiaru dla dx = 0.0000001 dla kolejno 1,2,4 wątków

Obraz zawierający tekst, zrzut ekranu, paragon, Czcionka

Opis wygenerowany automatycznie

Wnioski:

Wykorzystanie obliczeń równoległych pozwala zwiększenie wydajności oraz skrócenie czasu wykonywania złożonych obliczeniowo programów.

W przypadku obliczania sumy w programie pthreads\_suma dla tablicy o rozmiarze 100000000 można zauważyć wykonanie programu nie raz dwukrotnie szybciej, przy wykorzystaniu dla 2 wątków. W przypadku małej tablicy o rozmiarze 1000 program sekwencyjnie wykonuje się szybciej z tego względu, iż zarządzanie wątkami oraz uruchamianie wątków zajmuje dużo czasu w stosunku do obliczeń sekwencyjnych na tak małej tablicy, w tym przypadku obliczenia sekwencyjne okazują się szybsze, lecz jak widać na większym rozmiarze tablicy, użycie n wątków już skraca czas obliczeń około n-krotnie.

W przypadku obliczania całki w programie obliczanie\_calki widać, że zrównoleglenie również skraca czas obliczeń kilkukrotnie w przypadku niskiego dx (wysokości trapezu), rzędu 0,000001 ze względu na dużą ilość obliczeń do wykonania. W przypadku wyższych trapezów, przy czym mniejszej ilości obliczeń, program wykonywany sekwencyjnie ma krótszy czas wykonania. Działa to na podobnej zasadzie jak przy programie pthreads\_suma, ponieważ zarządzanie wątkami oraz ich inicjowanie wymaga trochę dodatkowego czasu, który zwraca się w szybszym wykonaniu dużych obliczeń. Wynik całki jest zaokrąglany do podobnej nieomal identycznych wartości w przypadku sekwencyjnym oraz zrównoleglonym, ponieważ mamy tę samą liczbę iteracji oraz wysokość trapezu dx, różnice co najwyżej mogą wynikać z powodu maszyny, na której wykonywany jest program. Zwiększenie liczby trapezów (zmniejszenie dx) pozwala nam na uzyskanie dokładniejszych wyników całkowania (np. ok. 1,99 przy dx = 0.1 oraz ok. 1,999999999999 przy dx = 0,000001), odbija się to na czasie wykonywania programu.