Przetwarzanie równoległe – sprawozdanie

Mateusz Kreczmer 151736

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Numer programu | Liczba wątków-procesów | Czas przetwarzania  Równoległego [s] | Czas procesorów  [s] | Przyspieszenie |
| P2 | 4 | 0.054000 | 0.054000 | 2.167 |
|  | 8 | 0.048000 | 0.048000 | 2.500 |
| P3 | 4 | 3.113000 | 3.113000 | 0.042 |
|  | 8 | 4.240000 | 4.240000 | 0.028 |
| P4 | 4 | 0.049000 | 0.049000 | 2.082 |
|  | 8 | 0.047000 | 0.047000 | 2.277 |
| P5 | 4 | 0.054000 | 0.054000 | 2.000 |
|  | 8 | 0.042000 | 0.043000 | 2.605 |
| P6 | 4 | 2.393000 | 2.393000 | 0.061 |
|  | 8 | 1.883000 | 1.883000 | 0.098 |

Obliczenia zostały wykonane na procesorze posiadającym 4 rdzenie.

Liczba iteracji pętli for wynosi w wersjach kodu PI2-PI6 wynosi 100 000 000. Natomiast w przypadku PI7 wynosi ona 10 000 000.

1. Wersja druga kodu – PI2

zmienne prywatne – x, i, num\_steps, steps

zmienne współdzielone – pi, sum

Współdzielenie dotyczy zarówno odczytu, jak i zapisu dla zmiennej *‘sum’*. Każdy wątek zapisuje wartość do *‘sum’*, co prowadzi do wyścigów danych.

Wynik obliczeń jest niepoprawny ze względu na wyścigi danych w przypadku współdzielonej zmiennej *‘sum’*. Kilka wątków równocześnie próbuje aktualizować wartość sum, co prowadzi do utraty danych i nieprzewidywalnych wyników.

Liczba dostępów do pamięci wzrasta ze względu na współdzielenie zmiennej *‘sum’* między wątkami głównie z powodu potrzeby synchronizacji i koordynacji dostępu do tej zmiennej. Gdy wiele wątków równocześnie próbuje odczytywać i aktualizować wartość zmiennej *‘sum’*, muszą one komunikować się ze sobą w celu zapewnienia spójności danych. Proces ten wymaga częstych dostępów do pamięci, ponieważ wątki muszą czytać i zapisywać dane w pamięci współdzielonej.

1. Wersja trzecia kodu – PI3

Wyniki obliczeń tym razem są poprawne, ponieważ operacje zapisu do ‘*sum*’ są wykonywane atomowo za pomocą dyrektywy ‘*atomic*’.

W związku w powyższym liczba dostępów do pamięci wzrosła w tej wersji kodu w porównaniu do poprzedniej, ponieważ zastosowanie dyrektywy ‘*atomic*’ wymaga dodatkowych operacji synchronizacyjnych, co prowadzi do większej ilości dostępów do pamięci.

Możemy także zaobserwować, że zwiększenie liczby wątków wprowadziło dodatkowy narzut synchronizacyjny i komunikacyjny, co zwiększyło czas przetwarzania. Ma to sens, ponieważ dyrektywa ‘*atomic*’ wymusza na wątkach koordynację dostępu do zmiennej współdzielonej.

1. Wersja czwarta kodu – PI4

Każdy wątek używa własnej prywatnej zmiennej *‘sum\_private’* do obliczenia sumy częściowej. Zapewnia to, że każdy wątek niezależnie przetwarza swoją część danych.

Po zakończeniu pracy wszystkich wątków, ich wyniki są atomowo dodawane do zmiennej globalnej *‘pi\_total’*, zapewniając poprawne scalenie wyników.

Liczba dostępów do pamięci jest nadal dość wysoka ze względu na konieczność synchronizacji i komunikacji między wątkami podczas dodawania wyników do zmiennej globalnej. Jednakże, zastosowanie zmiennych prywatnych zmniejsza konkurencję o dostęp do tej zmiennej, co zdecydowanie zmniejsza narzut synchronizacyjny w porównaniu do poprzedniej wersji kodu.

1. Wersja piąta kodu – PI5

Używając klauzuli *‘reduction(+:pi)’* przy dyrektywie *‘parallel for’*, każdy wątek automatycznie sumuje swoje lokalne sumy częściowe do zmiennej pi, co eliminuje potrzebę ręcznego scalania wyników.

Wykorzystanie klauzuli *‘reduction’* znacznie zmniejsza liczbę dostępów do pamięci w porównaniu do użycia dyrektywy *‘atomic’* lub zmiennej globalnej do zapisu wyników. Wartości są scalane lokalnie w każdym wątku, co minimalizuje konkurencję o dostęp do pamięci współdzielonej.

1. Wersja szósta kodu – PI6

Wyniki obliczeń są poprawne, ale czas przetwarzania jest wyższy niż w przypadku poprzednich wersji kodu. Spowodowane jest to występowaniem zjawiska *‘false sharing’*, co oznacza, że różne wątki modyfikują sąsiednie elementy tablicy. Prowadzi do unieważniania kopii linii pamięci podręcznej, co z kolei zwiększa czas przetwarzania.

Przetwarzanie wersji PI6 cechuje się gorszą wydajnością niż wersje PI4 i PI5 z powodu ww. zjawiska i konieczności sprowadzania do procesora aktualnych wersji linii danych pamięci podręcznej.

1. Wersja siódma kodu – PI7

W tej wersji dla każdej iteracji wykonujemy obliczenia na dwóch sąsiednich słowach tablicy. Przykładowo, w pierwszej iteracji oba wątki pracują na elementach tablicy tab[0] i tab[1], w drugiej iteracji na tab[1] i tab[2], itd. Dzięki temu, mamy kontrolę nad używanymi danymi i możemy zaobserwować, jak czas przetwarzania różni się między kolejnymi iteracjami.

Dzięki temu, że każdy wątek pracuje na dwóch sąsiednich słowach tablicy, ale nie na tych samych słowach, zapobiegamy występowaniu zjawiska *‘false sharingu’*. Słowa, na których operuje każdy wątek, są od siebie oddalone o co najmniej jedno słowo tablicy. W rezultacie, mimo że wątki mogą dzielić tę samą linię pamięci podręcznej procesora, to nie następuje konflikt między nimi, ponieważ operują na różnych danych.

Dlatego, w każdej iteracji, czas przetwarzania jest cyklicznie krótszy co 8 iterację, ponieważ wątki operują na danych, które znajdują się na różnych liniach pamięci podręcznej procesora, co minimalizuje występowanie false sharingu.

