Pomiar prędkości przetwarzania w funkcji użytej liczby wątków/procesorów dla różnych wersji kodu – obserwacja kosztów współdzielenia danych między wątkami. System komputera wielordzeniowego z pamięcią współdzieloną, środowisko Windows/Visual Studio lub Linux

Specyfikacja

- procesor: Intel Core i7 8550U
- ilość rdzeni: 4
- ilość wątków logicznych: 8
- Hyper-Threading: tak
- System Operacyjny: Windows 10 20H2

Wersja 1 (PI1) - przetwarzanie sekwencyjne

- czas obliczeń: 1 090ms

Wersja 2 (PI2) - proste zrównoleglenie

- czas obliczeń:
  - 2 wątki: 1 425 ms
  - 4 wątki: 1 862 ms
  - 8 wątków: 3 490 ms
- lokalność:
  - zmienne lokalne: `i`
  - zmienne współdzielone: *wszystkie pozostałe*
- przyspieszenie:
  - 0.76x *(w najlepszym przypadku)*

Czas przetwarzania w wersji równoległej jest dłuższy względem wersji szeregowej. Spodziewany (błędnie) wzrost prędkości przetwarzania nie nastąpił bez względu na ilość wątków logicznych. Przyczyną takiego zachowania jest wielokrotne uniważnianie liń pamięci w czasie przetwarzania ze względu na współdzielenie zmiennych, np. `x`, albo `sum`. Ze względu na występujący wyścig w dostępie do zmiennej `sum` wynik przetwarzania jest niepoprawny.

Wersja 3 (PI3) - atomic

- czas obliczeń:
  - 2 wątki: 10 274 ms
  - 4 wątki: 23 156 ms
  - 8 wątków: 75 109 ms

- lokalność:
  - bez zmian
- przyspieszenie:
  - 0.1x *(w najlepszym przypadku)*

Zastosowanie klauzuli `#pragma omp atomic` spowodowało, że obliczenia równoległe kończą się zwróceniem poprawnego wyniku. Niestety kosztem wymuszenia atomowości uaktualnienia wartośći zmiennej `sum` jest znaczne wydłużenie czasu wyknonania programu względem wersji sekwencyjnej. Głownym powodem jest potrzeba każdorazowego zakładania blokady na zmienną `sum`, co w przypadku proponowanego programu sprowadza go do wersji sekwencyjnej. Dalsze pogorszenie czasu przetwarzania wiążę się z potrzebą pobierania do pamięci cache procesora nowych wartości zmiennej `sum`, w większości przypadków, kiedy chcemy ją aktualizować. Zmienna jest współdzielona przez wszystkie wątki, a każdy wątek w każdym wykonaniu pętli ją aktualizuje, co powoduje unieważnienie lokalnych kopii tej zmiennej dla wątka/rdzenia/procesora.

Wersja 4 (PI4) - lokalne zmienne

- czas obliczeń:
  - 2 wątki: 1 045 ms
  - 4 wątki: 1 081 ms
  - 8 wątków: 1 125 ms
- lokalność:
  - zmienne lokalne: `i`, `suml`, `x`
  - zmienne współdzielone: *wszystkie pozostałe*
- przyspieszenie:
  - 1.04x *(w najlepszym przypadku)*

W wyniku "lokalizacji" zmiennych czas przetwarzania równoległego, poraz pierwszy okazał się krótszy od czas przetwarzania sekwencyjnego. Dodanie zmiennych lokalnych spowodowało, że nie ma już potrzby każdorazowego uniważniania lińpamięci. Od teraz może to wystąpić tylko w momencie dodawania lokalnej sumy do sumy globalnej, jednak takich dodawań jest znacznie mniej niż wcześniej, co przyczyniło się do wrostu tempa przetwarzania.

Wersja 5 (PI5) - redukcja

Jedyna zmiana zachodząca w kodzie to dodanie nowej części `reduction(+:sum)` do istniejącej dyrektywy `#pragma omp parallel for`. Nie powoduje to istotnych zmian w generowanym kodzie, dlatego wszystkie wyniki z Wersji 4 pozostają aktualne.

Wersja 6 (PI6) - tablica (false sharing)

- czas obliczeń:

  - 2 wątki: 663 ms
  - 4 wątki: 782 ms
  - 8 wątków: 844 ms
- lokalność:
  - zmienne lokalne: `i`, `x`
  - zmienne współdzielone: *wszystkie pozostałe*
- przyspieszenie:
  - *(w najlepszym przypadlu)*

Po zastosowaniu tablicy do przechowywania danych czas przetwarzania dla wszystkich ilości wątków jest większy od czasu przetwarzania sekwencyjnego. Wynik przetwarzania jest poprawny. Fakt wydłużenia czasu obliczeń można tłumaczyć zjawiskiem false sharing - linie adresowe dla poszczególnych wątków procesora *nachodzą na siebie* a przez to są unieważniane i wymagają ponownego pobrania do procesora.

Wersja 7 (PI7) - tablica (pomiar długości lini pamięci procesora)

Wersja 7 programu jest rozszerzeniem Wersji 6, tak aby sprawdzić jaka jest szerokość lini danych pamięci cache procesora. Eksperyment zakłada wykorzystanie tylko dwóch wątków procesora. Każdy z wątków będzie zapisywał wyniki swoich obliczeń do osobnej komórki tablicy typu double (rozmiar 64 bity). W trakcie eksperymentu będziemy zmieniali adresy pod które zapisywane są dane w następujący sposób: *id_watku* + k, gdzie k jest pewną stałą zmieniającą się w każdej iteracji eksperymentu. W ten sposób sprawimy, że dla pewnych k - występujących z stałym okresem - czas wykonania programu znacząco spadnie. Różnica pomiędzy dwoma sąsiednimi k będzie długością lini adresowej pomniejszonej 64 razy - rozmiar zmiennej double.
Niestety eksperyment jest wrażliwy na warunki platformy sprzętowej na jakiej jest uruchamiany. Jednym z występujących problemów jest uruchamianie programu na jednym rdzeniu procesora w technologi HT. Powoduje to, że zmienne cache procesora są współdzielone i nie ulegają unieważnieniu - nie widać dla jakich k uzyskujemy przyspieszenie, ponieważ wszystkie czasy są takie same.
Dla mojego przypadku przyspieszenie uzyskujemy dla k=$\{1,3,5,7, \ldots\}$, co wskazuje na różnicę 1 co oznacza, że długość lini danych pamięci cache jest równa 64 bity co jest zgodne z dokumentacją techniczną procesora użytego do eksperymentu.