Przemysław Szymoniak

Przetwarzanie współbieżne. Programowanie równoległe i rozproszone.

Sprawozdanie z laboratorium 10.
Celem laboratorium było zwiększenie umiejętności pisania programów równoległych w środowisku OpenMP

W ramach zajęć zrealizowałem następujące kroki:

- Pobrałem i rozpakowałem wszystkie wymagane pliki i uruchomiłem program.
- Dodanie wymaganych barier, sekcji krytycznych i operacji atomowych:

```
#pragma omp parallel default(none) shared(a_shared, e_atomic) private(b_private) firstprivate(c_firstprivate ) num_threads(liczba_watkow)
  {
    int i;
    int d_local_private;


    d_local_private = a_shared + c_firstprivate;
    f_threadprivate = omp_get_thread_num();
    #pragma omp barrier
    for(i=0;i<10;i++){
      #pragma omp critical
1.    //Wyeliminowana zależności zasobów
      a_shared ++;
    }

    for(i=0;i<10;i++){
      c_firstprivate += omp_get_thread_num();
    }

    for(i=0;i<10;i++){
      #pragma omp atomic
2.    //Wyeliminowana zależności zasobów
      e_atomic+=omp_get_thread_num();
    }


    #pragma omp barrier
    #pragma omp critical
    {

      printf("\nw obszarze równoległym: aktualna liczba watkow %d, moj ID %d\n",
         omp_get_num_threads(), omp_get_thread_num());

      printf("\tb_private \t= %d\n", b_private);
      printf("\tc_firstprivate \t= %d\n", c_firstprivate);
      printf("\td_local_private = %d\n", d_local_private);
3.    //Wyeliminowana zaleznośćć danych(zapis po odczycie)
      printf("\ta_shared \t= %d\n", a_shared);
      printf("\te_atomic \t= %d\n", e_atomic);

    }
```

- Opis zależności:
  1. Zależności zostają wyeliminowane dzięki użyciu sekcji krytycznej dla **a_shared**:
     - Zależność zasobów powstaje kiedy więcej niż jeden wątek chce dokonać operacji na zasobie (w tym wypadku zapis zmiennej w tym samym momencie). Została wyeliminowana poprzez użycie sekcji krytycznej, nie pozwalającej na uzyskanie przez parę wątków dostępu w tym samym momencie.
     - Alternatywnie zamiast zależności zasobów można dopatrzeć się zależności wyjścia danych, ponieważ więcej niż jeden wątek w podobnym okresie czasu wykonuje operacji zapisu(zapis po zapisie).

  2. Zależności zostają wyeliminowane dzięki użyciu operacji atomowej dla **e_atomic**:
     - Zależność zasobów powstaje kiedy więcej niż jeden wątek chce dokonać operacji na zasobie (w tym wypadku zapis zmiennej w tym samym momencie). Została wyeliminowana poprzez użycie sekcji krytycznej, nie pozwalającej na uzyskanie przez parę wątków dostępu w tym samym momencie.
     - Alternatywnie zamiast zależności zasobów można dopatrzeć się zależności wyjścia danych, ponieważ więcej niż jeden wątek w podobnym okresie czasu wykonuje operacji zapisu(zapis po zapisie).

  3. Zależność eliminujemy przez wykorzystanie bariery, a blokowy zapis danych wątku przez sekcje krytyczną
     - Anty-zależność danych powstała w wyniku różnego czasu wykonywania poszczególnych operacji dla różnych wątków. Część wątków jeszcze oblicza **e_atomic** oraz **a_shared** w czasie gdy my zaczynamy je już wypisywać. W ten sposób dla każdego wątku wypiszemy troszkę inne dane a chcemy identyczne.

Wnioski:

- W kodzie mogą występować różne zależności. Aby wyeliminować jedne wymagane zmiany są kosmetyczne przy innych może być konieczność większych zmian w algorytmie tak jak w przypadku rzeczywistych zależności danych.
- W celu eliminacji zależności z pomocą przychodzą **sekcja krytyczna**, **operacja atomowa** oraz **bariera**.
- Dane współdzielone miedzy wątkami dla każdego z nich powinny posiadać taką samą wartość.
- Przy testowaniu warto zwrócić uwagę, że część problemów nie występuje przy wykorzystaniu małej liczby wątków. Z powodu dużej mocy obliczeniowej współczesnych komputerów, wymagana jest większa ilość aby móc zaobserwować niektóre problemy.