OpenMP – podstawy

Koszty współdzielenia danych w systemach równoległych Linux/Windows

UWAGA:

Pytania dotyczące zadania proszę przesyłać przez email na adres prowadzącego zajęcia, odpowiedzi znajdą się na stronie przedmiotu dla wykorzystania przez wszystkich.

Zadania-cele:

- uruchomienie prostych programów przy wykorzystaniu określonej wprost liczby wątków przetwarzania

- zastosowanie dyrektyw #pragma omp parallel, for, atomic, reduction

- zastosowanie funkcji omp: `omp_set_num_threads(),omp_get_thread_num()`

-pomiar czasu przetwarzania równoległego

- obserwacja kosztów współdzielenia danych

Zadanie 1 – prosty program w Open_MP (hello_world.c)

Kompilacja i uruchomienie Linux:

- gcc –fopenmp plikwe –o plikwyj
- ./plik wyj
- Celem przełącznika –fopenmp jest uruchomienie kompilacji z uwzględnieniem dyrektyw OpenMP

Przygotowanie, kompilacja i uruchomienie projektu w Windows/Visual_studio:

- File/New/Project
- Project types/VisualC++/Win32/Templates/Win32/Templates/Console Application
- W **Win32Application Wizard** Application Settings/Additional Options/Empty Project
- W **Solution Explorer** Source Files (prawy klawisz myszy) Add/ New Item - Wprowadź kod: np. HelloWords.cpp

Włączenie generacji kodu zgodnie z dyrektywami OPen MP:

- Project > Properties
- C/C++ > Language > OpenMP Support - wybrać Yes
- Prosze wybrać wersję kodu generowanego: Release
- Build/Build Solution (uruchamianie kodu do testów czasowych w trybie Release)
- Debug/Start without Debugging
  Przekazanie do kompilatora informacji o potrzebie interpretacji dyrektyw OpenMP – Project/Properties/Language/C++/OpenMP suport Yes

Kroki pracy z kodem 1:

1. Wprowadzenie pliku nagłówkowego umożliwiającego pracę z funkcjami OpenMP:

- `#include <omp.h>`

2. Wprowadzenie do kodu dyrektywy zrównoleglającej przetwarzanie w **kolejnym po dyrektywie bloku kodu**:

- `#pragma omp parallel`

3. Zmienne prywatne i zmienne współdzielone wątków, obserwacja efektu wyścigu występującego w kodzie podczas modyfikacji zmiennych współdzielonych.

4. Ustalenie liczby uruchamianych wątków

- `omp_set_num_threads(liczba_uruchamianych_wątków)`

5. Uzyskanie informacji o wątku - źródle komunikatów pojawiających się na konsoli

- zastosowanie funkcji: `omp_get_thread_num()`

Obserwacja kolejności pojawiających się komunikatów świadczących o pracy równoległej przy użyciu wielu wątków przetwarzania (w zależności od określonej liczby wątków) – ewentualny niedeterminizm przetwarzania.

Użycie dyrektywy podziału pracy w ramach pętli for:

- #pragma omp for
- Obserwacja numerów iteracji wykonywanych przez poszczególne wątki (określone identyfikatorem)

- Analiza efektów różnych wariantów dyrektywy podziału pracy **#pragma omp for** (porównaj wykład lub dokumentację OPENMP)
    - podział pracy domyślny
    - podział pracy statyczny blokowy
    - podział pracy statyczny cykliczny
    - podziłał pracy dynamiczny domyślny i sparametryzowany
    - podział dynamiczny - sterowany

Zadanie 2 - 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

1. **Wersja pierwsza kodu PI1** - Wykorzystać **sekwencyjny** kod wyznaczenia wartości liczby pi metodą całkowania (Serial_pi.cpp). Całkowanie dotyczy funkcji okręgu o promieniu równym 1 umieszczonego w początku układu współrzędnych. Funkcja jest całkowana w zakresie argumentów od 0 do 1, całkowanie dotyczy obszaru ¼ koła o promieniu 1 położonego nad osią X. Obszar ten ma powierzchnię pi/4. Szukana liczba pi jest wyznaczona jako iloczyn wyznaczonego pola ¼ koła i liczby 4.

- Pomiaru czasu (Linux) pracy procesorów nad programem użytkownika (suma czasu pracy wszystkich użytych procesorów) za pomocą funkcji clock() (#include<time.h>), pomiar czasu w jednostkach będących częścią CLOCKS_PER_SEC sekundy
- Pomiaru **upływu czasu rzeczywistego** w ramach przetwarzania: funkcja omp_get_wtime() (#include <omp.h>) zwraca wartość typu double oznaczającą liczbę sekund, które upłynęły od arbitralnie wybranego momentu w przeszłości. Dwukrotne wywołanie funkcji pomiaru czasu i odjęcie otrzymanych wartości pozwala na określenie czasu, który upłynął między uruchomieniami funkcji (np. przed i po przetwarzaniu, którego czas jest mierzony).
- Liczba iteracji pętli – liczba kroków całkowania powinna umożliwić uzyskanie wyniku w czasie pojedynczych sekund. Kolejne eksperymenty (inne wersje kodu) wykonać dla tej samej liczby iteracji. W systemie Windows/Visual Studio należy wybrać generację kodu Release, w systemie Linux kompilator wprowadzić w tryb optymalizacji kodu przełącznikiem –O3

- W kolejnych punktach zadnia – praca z kolejnymi wersjami kodu:
    - Dla wersji równoległych uruchomić przetwarzanie przy użyciu liczby wątków równej:
        - liczbie procesorów logicznych
        - liczbie procesorów fizycznych
        - połowie liczby procesorów fizycznych
    - Dla każdej kolejnej wersji kodu równoległego PI2-PI6 należy:
        - porównać czas obliczeń z czasem wersji sekwencyjnej,
        - określić przyczyny wpływające na czasy przetwarzania różne/jednakowe z czasem obliczeń sekwencyjnych, uwzględnić liczbę wątków i procesorów uczestniczących w przetwarzaniu oraz spodziewany stopień lokalności dostępu do danych/zmiennych,
        - określić czy występuje i dlaczego? unieważnianie kopii linii pamięci podręcznej danych procesora przechowywanych w pamięciach podręcznych, czy jest to znaczące dla czasu przetwarzania?
        - dla systemu Linux porównać czas trwania obliczeń (wall clock) oraz czas użycia procesorów (możliwy do pomierzenia w systemie Linux) i wyjaśnić przyczyny uzyskanych wartości.

- obliczyć przyspieszenie przetwarzania w wyniku zastosowania równoległości czyli iloraz czasu przetwarzania sekwencyjnego i czasu przetwarzania badanej wersji równoległej.

2. **Wersja druga kodu - PI2** Zrównoleglić obliczenia wyznaczania liczby PI poprzez dyrektywę utworzenia regionu równoległego #pragma omp paralel i dyrektywę podziału pracy `#pragma omp for` w ramach pętli for. Przeanalizować które zmienne są prywatne, a które współdzielone (czy współdzielenie dotyczy odczytu czy zapisu), zaobserwować wynik obliczeń (jakie mogą być przyczyny niepoprawnego wyniku?) i zapisać czasy przetwarzania.
3. **Wersja trzecia kodu - PI3** Zrealizować ciąg dostępów (odczyt, zapis) do współdzielonej sumy w sposób niepodzielny:
   #pragma omp atomic
   sum+=4.0/(1.+ x*x);
   usunąć błędy współdzielenia pozostałych zmiennych, zaobserwować wynik obliczeń, zapisać czasy przetwarzania. Proszę sprawdzić w opisie standardu Open MP (np. wykład OpenMP) jakie są konsekwencje dla wynikowego kodu generowanego przez kompilator w odpowiedzi na umieszczenie w kodzie źródłowym dyrektywy **atomic.** W jaki sposób można na poziomie wątku zapewnić atomowość (niepodzielność) uaktualnienia zmiennej współdzielonej w systemie.
4. **Wersja czwarta kodu - PI4** Utworzyć i użyć zmienne prywatne każdego z wątków do zapamiętania wyniku lokalnej pracy oraz użyć dyrektywy #pragma omp atomic do zapewnienia poprawności scalenia w ramach zmiennej globalnej wyników pracy na lokalnej zmiennej - zaobserwować wynik obliczeń, zapisać czasy przetwarzania.
5. **Wersja piąta kodu - PI5** Zrealizować automatycznie scalenie wartości obliczanych w lokalnych sumach częściowych - klauzula `reduction`, zaobserwować wynik obliczeń i zapisać czas.
6. **Wersja szósta kodu - PI6** Zaimplementować sumy częściowe wyznaczane przez poszczególne wątki w ramach współdzielonej przez wątki tablicy zmiennych typu `double` – każdy wątek modyfikuje jedno słowo w tablicy zapisując w nim swoją sumę częściową – modyfikowane przez różne wątki słowa są sąsiednimi elementami tablicy. Zaobserwować wzrost czasu przetwarzania spowodowany przez fakt niezamierzonego/fałszywego współdzielenia danych (ang. false sharing). Zjawisko to polega na wielokrotnym, cyklicznym (wywołanym przez wielokrotne zapisy w różnych procesorach wartości do tej samej linii pamięci podręcznej procesora):
   - unieważnianiu powielonych linii pamięci w pp procesorów, które przestają być aktualne w wyniku zapisu danych do jednej z wielu kopii tej linii,
   - konieczności sprowadzenia do pamięci podręcznej procesora realizującego kod wątku (korzystającego z danych sąsiednich) aktualnej kopii danych z unieważnionej linii.

   Proszę zaobserwować wyniki i zapisać czas.
   Przetwarzania powinno cechować się gorszą wydajnością niż wersja PI4. Wynika to ze spodziewanego dla tej wersji kodu unieważniania i konieczności sprowadzania do procesora aktualnej wersji linii danych pamięci podręcznej. Unieważnianie to jednak może nie wystąpić z jednego z dwóch powodów:
   - przetwarzania wszystkich wątków na jednym fizycznym procesorze (2 procesory logiczne w technologii Hypertreading – jedna pamięć podręczna i jedna kopia linii pamięci dostępna przez wszystkie wątki),
   - optymalizacji kodu przeprowadzanej przez kompilator. W wyniku tej optymalizacji może powstać kod wynikowy, w którym wartości zmiennych nie są zapisywane w pamięci, lecz pamiętane w rejestrach procesora do momentu zakończenia przetwarzania bloku kodu , w którym są wykorzystywane. Na wywołanie niezamierzonego współdzielenia można wpłynąć deklarując zmienną (której wartości mają być przechowywane w pamięci) przy użyciu dyrektywy volatile np. volatile double tab[n];

7. **Wersja siódma kodu - PI7** Eksperyment z rozszerzoną wersją koduPI6 – celem eksperymentu jest wyznaczenie długości linii danych pamięci podręcznej procesora w oparciu o niezamierzone/fałszywe współdzielenie.

Koncepcja eksperymentu polega na znalezieniu adresów w pamięci odpowiadających sąsiednim końcom linii pamięci podręcznej – wyznaczona odległość miedzy dwoma sąsiednimi końcami linii pp jest długością linii. Różnica adresów jest różnicą położenia wykorzystywanych słów zmiennej tablicowej zajmującej ciągły obszar pamięci. Odległość między dwoma adresami słów zmiennej tablicowej wynika z rozmiaru tych słów (jeżeli korzystamy z elementu tab[5] i elementu tab[8] zmiennej tablicowej tab[n] typu double (8 bajtów) to niewykorzystany obszar pamięci (między tab[5] i tab[8] ma wielkość 16 bajtów).

W ramach tego eksperymentu wykorzystujemy:

- jedno uruchomienie kodu
- jedną zmienną tablicową zajmującą ok. 50 słów 8 bajtowych czyli kolejnych 400 bajtów pamięci
- dwa wątki pracujące (domyślnie) na dwóch procesorach fizycznych z różnymi pamięciami podręcznymi procesora (dla wywołania unieważnienia kopii powielonych linii pp).
- zakładamy, że słowo zmiennej będzie wyrównane do adresu początkowego linii pamięci (jeśli by tak nie było należy zrezygnować z wykorzystania sąsiednich słów, a użyć za każdym razem dwóch słów tablicy oddzielonych słowem niewykorzystanym).
- Realizujemy równolegle w kodzie obu wątków wiele odczytów i **zapisów** do sąsiednich słów tej samej tablicy przechowujących sumy częściowe algorytmu obliczania liczby pi.
- Przebieg eksperymentu:
    - Jedno uruchomienie kodu przy użyciu 2 wątków (tablica deklarowana z volatile).
    - W ramach tego uruchomienia wątki wielokrotnie obliczają liczbę pi, przy użyciu za każdym razem innej pary słów z tablicy: tab[0] i tab[1]; tab[1]i tab[]2]; tab[2]i tab[3]; aż do przejścia przez całą tablicę.
    - Dla każdej iteracji przy użyciu kolejnej pary słów należy wyświetlić czas obliczeń wraz z numerem iteracji odpowiadającej użytym słowom – elementom tablicy
    - Pracując przy użyciu 2 wątków na dwóch słowach pamięci z jednej tablicy możemy uzyskać różne czasy przetwarzania zależnie od tego czy słowa użyte znajdują się w jednej czy dwóch liniach. Czas krótszy wystąpi wtedy gdy praca 2 wątków przebiega przy użyciu 2 różnych linii, nie ma wtedy unieważnienia kopii linii pamięci podręcznej procesora. Czas krótszy określa zatem położenie jednego z końców linii. Znalezienie sąsiedniego miejsca w pamięci (poprzez wykorzystanie kolejno sąsiednich słów tablicy), któremu odpowiada krótszy czas obliczeń pozwala na znalezienie położenia kolejnego końca linii pamięci podręcznej. Odległość między kolejnymi końcami odpowiada poszukiwanej długości linii pamięci podręcznej procesora. Krótsze przetwarzanie będzie się powtarzać cyklicznie tak jak cyklicznie występują końce linii. Długość linii będzie podana wzorem zależnym od występujących w cyklu liczby iteracji o dłuższym czasie przetwarzania, liczby iteracji o krótszym czasie przetwarzania oraz wielkości słowa w pamięci.
    - Możliwe przyczyny braku powodzenia eksperymentu to niespełnione warunki konieczne: ciągły obszar danych – jedna tablica przez wszystkie iteracji obliczeń, dwa wątki przetwarzania, dwa procesory fizyczne – dwie pamięci podręczne procesora, zapisy do pamięci wywołujące niezamierzone współdzielenie (volatile), brak wyrównania słowa użytej tablicy do brzegu linii.

Opracowanie wyników eksperymentu to:

1. Określenie wykorzystywanego systemu równoległego: procesor, system operacyjny, kompilator
2. Tabela zawierająca dla kodów PI1 - PI6:
    - wyniki upływającego czasu rzeczywistego podczas obliczeń,
    - współczynnik przyspieszenia obliczeń dla uruchomień równoległych kodu i określonej liczby wątków (iloraz: czas sekwencyjny/czas równoległy)
    - uzasadnienie dla uzyskanej wielkości przyspieszenia – dlaczego czas przetwarzania równoległego jest dłuższy/ krótszy od czasu przetwarzania sekwencyjnego – proszę w wyjaśnieniach skorzystać z wykładu o zagadnieniach dotyczących współdzielenia pamięci w systemach równoległych i OpenMP.

3. Określenie długości linii wg przygotowanego wzoru i na podstawie wyników eksperymentu z kodem PI7 oraz uzasadnienie
4. Opisanie własnymi słowami przebiegu eksperymentu wyznaczenia długości linii pp procesora, uzasadnienie poprawności metody i wyniku, opis ewentualnych wątpliwości do wyników i trudności napotkanych podczas realizacji zadania.

Indywidualne opracowanie wyników powyższego zadania - plik *.pdf oraz pliki **źródłowe** zawierające poszczególne wersje kodu źródłowego PI2-PI7 proszę przygotować w postaci spakowanej *.zip proszę złożyć w ramach kursu Przetwarzanie równoległe systemu ekursy.put.poznan.pl w terminie miesiąca od PIERWSZYCH zajęć laboratoryjnej grupy, do której Pa
Nazwy plików: 1PRnrindeksu.pdf 1PRnrindeksu.zip gdzie nr indeksu jest numerem indeksu autora.

Materiały pomocnicze: wykłady: pamięć w systemach wieloprocesorowych z pamięcią współdzieloną, wykład/dokumentacja OpenMP.

Przygotowano: 2.03.2021