OpenMP – podstawy

Koszty współdzielenia danych

Laboratorium 2.6.22 (Intel)

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()`

–zastosowanie funkcji Windows API: `SetThreadAffinityMask()`, `GetCurrentThread()`

-pomiar czasu przetwarzania równoległego

- obserwacja kosztów współdzielenia danych

Proszę skorzystać z mteriałów pomcniczych do wykładu pt. Open MP

Zadanie 1 – prosty program w Open_MP

Utwórzenie projektu konsolowego Microsoft 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

Dostosowanie projektu do wykorzystania z kompilatorem Intel C++ i z Open MP:

- **Włączenie generacji kodu zgodnie z dyrektywami OPen MP:**
  - View/Solution Explorer
  - Wybrać projekt prawym klawiszem a dalej w pojawiającym się menu Project > Properties, następnie w oknie Property Pages wybrać:
    - C/C++ > Language > OpenMP Support wybrać Yes
  - **Prosze wybrać wersję kodu generowanego: Release**

Kompilacja i uruchomienie:

- Debug/Start without Debugging

Wprowadzenie nagłówka umożliwiającego pracę z funkcjami OpenMP:

- `#include <omp.h>`

Wprowadenie do kodu dyrektywy zrównoleglającej przetwarzanie w **kolejnym po dyrektywie bloku kodu** :

- `#pragma omp parallel`

Ustalenie liczby uruchamianych wątków

- `omp_set_num_threads(liczba_uruchamianych_wątków)`

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.

Zadanie 2 -

A) Zastosowanie funkcji SetThreadAffinityMask do ustalenia powinowactwa wątków do procesorów logicznych.

Udostępnienie funkcji Windows application programming interface (API)

```
#include <windows.h>
```

Uzyskanie uchwytu do bieżącego wątku:

```
HANDLE thread_uchwyt=GetCurrentThread();
```

Ustalenie maski powinowactwa dla przydziału wątków do procesorów modulo liczba procesorów (przykład poniżej):

```
int th_id=omp_get_thread_num();
//otrzymanie własnego identyfikatora

DWORD_PTR mask = (1 << (th_id % liczba_procesorów ));
//określenie maski dla przetwarzania wątku wyłącznie na jednym procesorze
przydzielanym modulo liczba procesorów

DWORD_PTR result = SetThreadAffinityMask(thread_uchwyt,mask);
//przekazanie do systemu operacyjnego maski powinowactwa

if (result==0) printf("blad SetThreadAffnityMask \n");
else
   {
   printf("maska poprzednia dla watku %d : %d\n",th_id,result);
   printf("maska nowa dla watku %d : %d\n",th_id,SetThreadAffinityMask(
thread_uchwyt, mask ));
   }
//sprawdzenie poprawności ustlenia maski powinowactwa
```

Zastosowanie maski powinowactwa dla systemu niejednorodnwgo (hyperthreading), badania lokalizacji procesora logicznego w strukturze systemu wieloprocesorowego współdzielnie lub nie fizycznego rdzenia.

B) Pomiar czasu przetwarzania z dużą rozdzielczością (cyklu procesora) - zastosowanie instrukcji rdtsc

Przygotowanie w kodzie:

- Ustalenie powinowactwa wątku (1 do 1) mierzącego czas:
- Wprowadzenie linii kodu pomiarowego:

```
#include <intrin.h>
#pragma intrinsic(__rdtsc)
unsigned long long a1,a2;
a1 = __rdtsc();
operacja której czas jest mierzony
a2 = __rdtsc();
printf("Czas przetwarzania =%llu ns \n", (a2-a1)/3);
//przeliczenie cykli procesora na nanosekundy przy uwzględnieniu częstotliwości taktowania procesora
// w przykładzie testowany procesor ma czestotliwosc procesora 3 GHz stad wynika zastosowany w kodzie przelicznik
```

Zadanie 3 - pomiar prędkości przetwarzania w funkcji użytej liczby wątków/procesorów dla różnych wersji kodu – obserwacja kosztów współdzielenia danych między wątkami.

1. Wykorzystać sekwencyjny kod wyznaczenia wartości liczby pi metodą całkowania (Serial_pi.cpp).
2. Dokonać pomiaru czasu przetwarzania za pomocą funkcji clock() (#include<time.h> , pomiar czasu w milisekundach).
3. Zrównoleglić obliczenia wyznaczania liczby PI poprzez dyrektywę podziału pracy `#pragma omp for`, przeanalizować które zmienne są prywatne, a które współdzielone, zaobserwować wynik obliczeń (jakie mogą być przyczyny niepoprawnego wyniku ?) i zapisać czas i liczbę iteracji.
4. Zrealizować ciąg dostępów do współdzielonej sumy w sposób niepodzielny `#pragma omp atomic` – **wersja pierwsza kodu** PI1, zaobserwować wynik obliczeń, zapisać czas i liczbę iteracji, jaką wielkość przyspieszenia obliczeń zaobserwowano?

5. Zrealizować scalenie wartości w lokalnych sumach częściowych klauzula `reduction` - **wersja druga kodu** PI, zaobserwować wynik obliczeń i zapisać czas i liczbę iteracji, jaką wielkość przyspieszenia obliczeń zaobserwowano?

6. Zaimplementować sumy częściowe w ramach współdzielonej tablicy zmiennych typu `double`, **wersja trzecia kodu** PI3.

Każdy wątek modyfikuje jeden element tablicy zapisując w nim swoją sumę częściową – modyfikowane elementy są sąsiednimi elementami tablicy.
Zaobserwować zjawisko „fałszywego” współdzielenia – ang. false sharing – polegające na dostępie - modyfikacji przez różne wątki nie tej samej lokacji danych, lecz lokacji sąsiednich (w tym celu tablica). Jeżeli dwa wątki są przetwarzane na różnych procesorach/rdzeniach korzystają z innych pamięci podręcznych. Zapis informacji w pamięci powoduje unieważnienie wszystkich kopii tej samej informacji w innych pamięciach. Ziarnistość informacji to linia pamięci podręcznej o wielkości 64 bajtów. Unieważnienie informacji w pamięci podręcznej w przypadku próby

dostępu do tych danych unieważnionych wymusza sprzętowo obsługiwane pobranie aktualnego bloku danych co związane jest z kosztem. Pobranie unieważnionych bloków danych w pp przed ich ponownym wykorzystaniem (odczyt lub zapis) realizowane wielokrotnie w kodzie spowalnia obliczenia. Proszę zmierzyć i zapisać czasy przetwarzania dla uruchomień kodu w różnych konfiguracjach liczby wątków i wykorzystywanych procesorów. Na podstawie czasu obliczeń proszę wyliczyć uzyskiwane przyspieszenia. **Wymusić zapis do pamięci** (nie w rejestrze procesora) za pomocą deklaracji tablicy przy użyciu dyrektywy **volatile**. Minimalizować liczbę zmian w kodzie dla poprawnego określenia kosztów „fałszywego współdzielenia". Wykorzystanie „fałszywego współdzielenia" dla określenie lokalizacji różnych procesorów fizycznych w masce powinowactwa.

7. Eksperyment z 3 wersją kodu – **wyznaczanie długości linii pp**
    a. Uruchamianie kodu przy użyciu 2 wątków uruchamianych **na różnych fizycznych procesorach** (tablica deklarowana z dyrektywą volatile).
    b. W kolejnych iteracjach eksperymentu obliczeniowego należy zmieniać położenie (względem stałego początku **raz zadeklarowanej tabicy**) 2 słów wykorzystywanych do zapisywania sum częściowych przez wątki. Proszę wyznaczyć i zapisać czasy obliczeń dla kolejnych 20 sąsiednich położeń względem początku wykorzystywanego obszaru tablicy – wyjaśnić uzyskane wyniki. Elementy niezbędne do realizacji eksperymentu: 2 wątki, 2 procesory/rdzenie fizyczne, zapisy do pamięci do sąsiednich lokacji, wielokrotne powtarzanie operacji dla umożliwienia poprawnego pomiaru czasu - kosztu obliczeń.

8. Zbadać (dla wersji kodu optymalnej pod względem czasu) wpływ powinowactwa wątków na czas przetwarzania (dla 4 i 8 wątków):
    - wątki dedykowane do procesorów 1 do 1,
    - wątki przydzielane po 2 do jednego procesora fizycznego,
    - wątki przydzielane po 1 do jednego procesora fizycznego.

    Z przeprowadzonych 3 krotnie uruchomień każdej testowanej wersji programu i rozmiaru problemu proszę wybrać do dalszych analiz i prezentacji czasy minimalne. Proszę przygotować również tabelę z czasami obliczeń dla poszczególnych wersji kodów z zapisaną liczbą wątków, sposobem przydziału wątków do procesorów, wielkością przyspieszenia przetwarzania równoległego poszczególnych wersji programu w stosunku do najkrótszego czasu przetwarzania sekwencyjnego uzyskanego dla jednakowego rozmiaru problemu (liczby kroków pętli).

    Zaliczenie ćwiczenia odbywa się poprzez rozmowę z prowadzącym zajęcia. Podczas rozmowy prezentowane są kody i wyniki, uruchamiane przykładowe wersje kodu i analizowane uzyskane wyniki. Proszę w czasie zaliczenia dysponować tabelą z wynikami pomiarów, obliczonych przyspieszeń oraz kodami pozwalającymi na uzyskanie prezentowanych wyników.

Zmodyfikowano: 4.03.2015