Змістовний модуль:

РОЗРОБКА ПАРАЛЕЛЬНИХ АЛГОРИТМІВ І ПРОГРАМ

Розділ 2.

РОЗРОБКА ПАРАЛЕЛЬНИХ ПРОГРАМ

Лекція 9. ТЕХНОЛОГІЯ OPEN MP. ВСТУП

ПИТАННЯ ДЛЯ ВИВЧЕННЯ

1. Загальна характеристика Open MP.
2. Включення режиму підтримки Open MP та перевірка успішності включення
3. Класифікація директив
4. Директиви для визначення паралельних ділянок і їх трансляція
5. Додаткові параметри директиви parallel

ЗАГАЛЬНА ХАРАКТЕРИСТИКА ЗАСОБІВ ДЛЯ РОЗРОБКИ || ПРОГРАМ

Для розробки C та C++ програм

1 Засоби вбудування коду для паралельного виконання на етапі компіляції:

компілятор

Open MP

2 Бібліотеки.

2.1 TBB – (Intel) – перевищує можливості Open MP

2.2 PPL – (Microsoft)

2.3 MPI

3 Мови

3.1 go (google)

3.2 erlang

ВБУДУВАННЯ КОДУ НА ЕТАПІ ТРАНСЛЯЦІЇ

Компілятор.

Properties->Code Generation ->Enable Parallel Code Generation->

Yes (/Qpar)

Приклад. Обчислити значення $y_i = \sin(x_i)$ $x_i = 0..4\pi$ (N = 10000000**)**

```
#define N10000000
#define M_PI       3.14159265358979323846
double y[N];
…
double h = 4 * M_PI / N, start, finish;
start = omp_get_wtime();
for (int i = 0; i < N; i++)       y[i] = sin(h* i);
finish = omp_get_wtime();
```

ЗАГАЛЬНА ХАРАКТЕРИСТИКА OPEN MP

Інтерфейс OPEN MP - стандарт програмування для однорідних багатопроцесорних систем з загальною пам'яттю. Прикладом такої системи є обчислювальна система із багатоядерним процесором.

Розробка стандарту - організація **OPEN MP ARB** (ARchitecture Board) - розробники SMP (Symmetric Multi Processor)- архітектури і програмного забезпечення (IBM, Intel, AMD).

Специфікації для мов *Fortran* і *C/C++* - жовтень 1997 року й жовтень 1998 року.

Сайт www.OPEN MP.org,

До 20 річчя – OPEN MP 5.0 (поки не викладено для завантаження)ю Остання версія 4.5 (листопад 2015 року). Документація за цією версією

http://www.openmp.org/mp-documents/openmp-4.5.pdf

Включає використання потоків, SIMD операцій і паралелізм для графічних карт

В курсі розглядається саме ця версія тільки для мов *C, C++.*

Технологія реалізована як в **UNIX** подібних системах, так і в сучасних **Windows** системах, починаючи із супер-еом і кінчаючи desktop

ПРИНЦИП ВИКОРИСТАННЯ OPEN MP

Принцип використання: в програму додаються спеціальні директиви. Ці директиви ігноруються, якщо компілятор не підтримує роботу з OPEN MP, або не включено відповідний режим. Директива ***#pragma*** мови *C++* обробляється саме так, тому вона використовується для цих цілей.

На початку виконання програми, як і звичайно, створюється первинний потік, що у рамках даної технології називається ***master* – потоком**.

Якщо задано директиву початку **паралельної секції**, то створюється необхідна кількість додаткових потоків, щоб забезпечити можливість паралельного виконання секції необхідну кількість раз. Для створення цих потоків компілятор формує необхідний код.

Потокова функція для цих потоків містить у собі оператори між директивою початку й кінця паралельної ділянки. Компілятор автоматично додає функції для синхронізації завершення потоків у крапці виходу з паралельної секції, код для знищення потоків.

Після завершення паралельного регіону виконується тільки *master* потік.

ВКЛЮЧЕННЯ РЕЖИМУ ПІДТРИМКИ OPEN MP

Для включення цього режиму необхідно:

1. Створити проект. Проект може бути будь-якого типу, у тому числі консольний.
2. Додати до проекту *C++* файл, якщо обрано пустий проект.
3. У поле Властивостей: *Project→ Properties→ <Ім'я проектуProperty→ Configuration Properties → C/C++ → Language → OPEN MP* вибрати *Yes*.

Увага!!!

Якщо Ви забули включити режим підтримки OPEN MP, то програма, в якій використовуються можливості цього режиму при трансляції, не буде видавати помилок, а просто буде працювати в послідовному режимі, тому настійно рекомендуємо перевіряти успішність включення цього режиму!

ПЕРЕВІРКА УСПІШНОСТІ ВКЛЮЧЕННЯ РЕЖИМУ ПІДТРИМКИ OPEN MP

Режим включений → компілятор визначає макрос **_OPENMP**.

Структура макроса: yyyymm - ціле дане, старші 4 цифри якого задають рік, а молодші місяць версії OPEN MP.

Чому важливо?

В деяких версіях включається тільки в режимі Release!

ПЕРЕВІРКА УСПІШНОСТІ ВКЛЮЧЕННЯ РЕЖИМУ ПІДТРИМКИ OPEN MP

Режим включений → компілятор визначає макрос **_OPENMP**.

Структура макроса: yyyymm - ціле дане, старші 4 цифри якого задають рік, а молодші місяць версії OPEN MP.

Таким чином, якщо макрос _OPENMP визначений, то можна зробити висновок про успішність включення режиму підтримки OPEN MP, а значення цієї змінної фактично визначає версію OPEN MP.

Код для перевірки успішності включення й визначення версії OPEN MP:

```
#ifdef _OPENMP
    _tprintf (TEXT("OPEN MP is Support. Version:Year %d, Month = %d\n"),
            _OPENMP/100, _OPENMP%100);
#else
            printf ("_OPEN MP is not defined.\n");
#endif
```

Для VS2015 Year = 2002, Month = 3. Ось чому ця реалізація не повністю підтримує можливості останніх версій.

КЛАСИФІКАЦІЯ ТА ЗАГАЛЬНИЙ ВИД ДИРЕКТИВ

Директиви діляться на:

- директиви визначення паралельних ділянок;
- директиви визначення класу пам'яті змінних;
- директиви для синхронізації.

Загальний вид директиви:

***#pragma omp* ім'я директиви** *[*Додаткові параметри*]*.

Додаткові параметри залежать від конкретної директиви. Додаткові параметри не обов'язкові, задають додаткову інформацію для директиви.

Якщо параметри задаються, то відокремлюються друг від друга комами.

Якщо директива не міститься в одному рядку, для її продовження використовується символ \ (як для макросу *#define*).

Увага!!!

Якщо при завданні директиви опустити ключове слово *omp*, то директива просто ігнорується, і замість паралельного виконання одержимо код, що буде виконуватися послідовно.

ДИРЕКТИВИ ДЛЯ ВИЗНАЧЕННЯ ПАРАЛЕЛЬНИХ ДІЛЯНОК. ЗАГАЛЬНА ХАРАКТЕРИСТИКА

Містять директиви:

- ***Parallel, for, sections, section, task, simd***

Директива *parallel* використовується для ділянки програми, що повинна виконуватись багаторазово: стільки разів, скільки створене потоків. Задається на початку паралельної ділянки незалежно від завдання інших директив.

Директива *for* використовується для паралельного виконання тіла циклу з відомою кількістю повторень. Ітерація циклу виконується одним потоком. Якщо кількість потоків менше кількості повторень циклу, то кілька ітерацій циклу виконується одним потоком. Розподіл навантаження між потоками визначається параметрами.

Директива *sections* використовується для завдання декількох ділянок програми, які можна виконувати паралельно. Кількість ділянок зазвичай співпадає с кількістю ядер. Усередині повинна бути задана одна або кілька секцій (**директива *section***). Кожна секція може виконуватися паралельно. Фактично одна секція відповідає одній потоковій функції

Директива *task* аналогічна директиві ***section ,*** *але декілька задач може виконуватись одним потоком в залежності від кількості ядер. (версія 3.0. та вище)*

Директива ***simd*** – щоб показати, що паралельне виконання забезпечується використанням simd операцій

ДИРЕКТИВИ ВИЗНАЧЕННЯ КЛАСУ ПАМ'ЯТІ ЗМІННИХ

Використовуються для визначення області видимості змінних. Змінні можуть бути локальними стосовно паралельної області (***private***) і загальними (***shared***), або загальними для потоку і різними для різних потоків(**threadprivate**).

Додаткові директиви цього класу використовуються для завдання правил за замовчуванням, особливостей ініціалізації й т.д.

ДИРЕКТИВИ СИНХРОНІЗАЦІЇ. ЗАГАЛЬНА ХАРАКТЕРИСТИКА

Дозволяють забезпечити атомарне виконання найпростіших операцій (директива ***atomic***), використання критичних секцій (директива ***critical***), виконання окремого коду тільки одним потоком (директиви ***master*** і ***single***), упорядкування виконання кодів в окремих потоках (***ordered***).

Поряд з директивами можна використовувати функції бібліотеки OPEN MP для синхронізації потоків.

ДИРЕКТИВА *PARALLEL*

Визначає початок і кінець ділянки програми для багаторазового виконання.

Загальний вид директиви:

***#pragma omp parallel* [Параметри]**

{

Блок операторів

}

Блок операторів виконується стільки разів, скільки потоків відповідає паралельній області.

Параметри:

if(Константний вираз);

default(shared | none);

firstprivate(Список змінних);

copyin(Список змінних);

proc_bind(master | close | spread)

num_threads(Константний вираз);

private(Список змінних);

shared(Список змінних);

reduction (Операція: Список змінних)

ПРИКЛАД ПРОГРАМИ З ПАРАЛЕЛЬНОЮ ДІЛЯНКОЮ

ВИВЕСТИ рядок "Hello, World\n“ в паралельному режимі

ПРИКЛАД ПРОГРАМИ З ПАРАЛЕЛЬНОЮ ДІЛЯНКОЮ

ВИВЕСТИ рядок "Hello, World\n“ в паралельному режимі

…

```
#ifdef _OPENMP
printf("Year  %d month = %d\n", _OPENMP /100,
    _OPENMP % 100);
#endif
#pragma omp parallel
{
printf("Hello, World\n");
}
```

ТРАНСЛЯЦІЯ ПАРАЛЕЛЬНОЇ СЕКЦІЇ

```
// Потокова функція
DWORD WINAPI ThreadFun (PVOID Par){
        Код для паралельного виконання
}
…
// Ділянка програми до паралельної секції
…
// Визначення кількості потоків (nThreads)
…
// Створення потоків
HANDLE *h = new HANDLE [nThreads-1];
for (int i = 0; i < nThreads-1; ++i) h [i] = CreateThread (0, 0, ThreadFun, 0, 0, 0);
ThreadFun ();
// Очікування завершення потоків
WaitForMultipleObject (nThreads, h, TRUE, INFINITE);
// Закриття потоків і звільнення пам'яті
for (int i = 0; i < nThreads - 1; ++i) CloseHandle (h [i]); delete [] h;
// Ділянка програми після паралельної секції
```

ДИРЕКТИВА *PARALLEL. ПАРАМЕТРИ*

Загальний вид директиви:

#pragma omp parallel [Параметри] {

Блок операторів

}

Блок операторів виконується стільки разів, скільки потоків відповідає паралельній області.

Параметри:

- ***if*(Константний вираз);**
- *num_threads*(Константний вираз);
- *default*(shared | none);
- *private*(Список змінних);
- *firstprivate*(Список змінних);
- *shared*(Список змінних);
- *copyin*(Список змінних);
- *reduction* (Операція: Список змінних)
- proc_bind(master | close | spread) – для останніх версій (affinite)

ВКЛЮЧЕННЯ – ВИКЛЮЧЕННЯ ПАРАЛЕЛЬНОГО ВИКОНАННЯ. ПАРАМЕТР *IF*

Вимикання. *Properties→ C/C++→Language* проекту виключити режим підтримки паралельного виконання (Open MP - off).

Недолік. Паралельне виконання відключається для всієї програми.

Як відключити тільки для ділянки програми???

Умовне відключення режиму паралельного виконання:

***if* (Вираз цілого типу)**

Вираз цілого типу = *true*, → паралельне виконання;

Вираз цілого типу = *false* → паралельного виконання немає.

#pragma omp parallel if (0)

паралельне виконання, задане директивою, вимикається.

***#pragma omp parallel if (1)*,**

паралельне виконання, задане директивою, вмикається.

Хай паралельне виконання має сенс, якщо кількість ітерацій більше 500.

#pragma omp parallel if (n > 500)

За замовченням *if (1).*

СПОСОБИ ВИЗНАЧЕННЯ КІЛЬКОСТІ ПОТОКІВ

1. По замовченню.
2. Використання параметру *num_threads.*
3. Використання функцій бібліотеки OPEN MP.
4. Використання змінних середовища OMP_NUM_THREAD і OMP_DYNAMIC.

По замовченню – число потоків дорівнює кількості ядер процесора

ВИКОРИСТАННЯ ПАРАМЕТРУ NUM_THREADS

Загальний вид параметра:

num_threads (вираз цілого типу).

Значення виразу визначає кількість потоків для даної паралельної ділянки.

Приклад. Задати *кількість потоків* = 16, і перевірити правильність її завдання.

```
int count = 0;
#pragma omp parallel if (1) num_threads (16)
 {
      printf("Hello, World\n"); }
 printf("Number of threads: %d\n", count);
```

Програма повинна видати відповідь *Number of threads*:16.

Увага!

Встановлена кількість потоків діє тільки на задану директиву ***parallel***.

ВИКОРИСТАННЯ ФУНКЦІЙ БІБЛІОТЕКИ OPEN MP

Ім'я	Вхідні дані	Вихідні Дані	Коментар
omp_set_num_threads	int	-	Встановлює кількість потоків. *nthreads-var [0]*
omp_get_num_procs() *omp_get_num_threads*	-	int	Кількість доступних ядер Кількість потоків для даної паралельної області
omp_get_max_threads	-	int	Максимальна кількість потоків Возвращает значение *nthreads-var [0]*
omp_get_thread_num	-	int	Номер поточного потоку
omp_set_dynamic *omp_get_dynamic*	int	-	Якщо a != 0, то кількість потоків визначається як по замовченню
omp_get_thread_limit	-	int	максимально припустима кількість потоків

ПРИКЛАДИ ВИКОРИСТАННЯ ФУНКЦІЙ

Приклад. Обчислити паралельно значення y_i = sin(x_i) x_i = 0..4π (N = 10000000**)**

ПРИКЛАДИ ВИКОРИСТАННЯ ФУНКЦІЙ

Приклад. Обчислити значення $y_i = \sin(x_i)$ $x_i = 0..4\pi$ (N = 10000000**)**

```
int threads = omp_get_num_procs();
printf("threads = %d\n", threads);
int Iters = N / threads;
start = omp_get_wtime();
#pragma omp parallel
{
int thread = omp_get_thread_num();
int begin = thread * Iters;
int end = begin + Iters;
for (int i = begin; i < end; i++)
y[i] = sin(h* i);
}
finish = omp_get_wtime();
```

ГАРАНТОВАНЕ ВСТАНОВЛЕННЯ КІЛЬКОСТІ ПОТОКІВ

Приклад 2. Безпечна функція

```
// Читаємо поточний режим
int dynamic = omp_get_dynamic();
// Встановлюємо потрібний режим, якщо
      необхідно
if (dynamic)
      omp_set_dynamic(0);
omp_set_num_threads(2);
…
// Повертаємо режим, який був с самого початку
omp_set_dynamic(dynamic);
```

ВИКОРИСТАННЯ ЗМІННИХ СЕРЕДОВИЩА OMP_NUM_THREAD І OMP_DYNAMIC

Встановлення змінної:

My computer → Properties → Advanced → Environment Variables → System variables і задаємо змінну зі своїм значенням (value).

Дія для OMP_NUM_THREAD:

If (value <0 || value >max)

реакція залежить від реалізації;

else

значення за замовчуванням = value;

Дія для OMP_DYNAMIC:

OMP_DYNAMIC=1 →nthreads = кількості ядер;

OMP_DYNAMIC=0 →nthreads залежить від установок;

ВИКОРИСТАННЯ ЗМІННИХ СЕРЕДОВИЩА. ПРИКЛАД

Нехай задане значення змінної середовища *OMP_NUM_THREADS* рівним 3.

Програма	Результат(кількість ядер=2)
#pragma omp parallel *{* *printf ("Hello\n");* *}*	Hello Hello Hello

ЗАГАЛЬНИЙ АЛГОРИТМ ВИЗНАЧЕННЯ КІЛЬКОСТІ ПОТОКІВ

```
if (omp_set_dynamic(1) задана || OMP_DYNAMIC ==1)
        кількість потоків дорівнює кількості ядер;
else {
        if (параметр num_threads (n))
            кількість потоків дорівнює n;
        else {
            if (omp_set_num_threads (m))
                        кількість потоків дорівнює m;
            else {
                        if (OMP_NUM_THREADS == k)
                                    кількість потоків дорівнює k;
                        else кількість потоків дорівнює = кількості ядер;
            }
        }
}
```

ПАРАЛЕЛЬНЕ ВИКОНАННЯ ЦИКЛУ

Нехай необхідно паралельно виконати цикл із заголовком

for (i = v1; i < v2; i += Step).

Кількість повторень циклу nFor:

nFor = (v2 - v1) / Step.

Хай кількість потоків:*nTthreads*, тоді кількість ітерацій на потік:

H = ⌐ *nFor / nTthreads* ¬

ПОТОКОВА ФУНКЦІЯ

```
// Кількість ітерацій циклу
#define      V1          0
#define      V2          8192
#define      Step        1
// Кількість потоків
#define nTthreads        2
typedef struct {
            int Start, Finish, Step;
            int x [V2];
}DATAS, *PDATAS;
DWORD WINAPI ThreadFun (PVOID Par){
      PDATAS pDatas = (PDATAS) Par;
      INT Start= pDatas ->Start, Finish= pDatas ->Finish, Step = pDatas ->Step;
      for (int i = Start; i < Finish; i+=Step){
            //          Тіло циклу
            pDatas ->x [i] = i * i;
      }
      _tprintf (_T("Start = %d Finish = %d Step = %d\n", Start, Finish, Step);
      return 0;
}
```

ГОЛОВНА ПРОГРАМА

```
HANDLE hThread [nTthreads - 1];
DATAS Datas [nTthreads ];
int i = 0, j;
// Кількість ітерацій
int nFor = (V2 - V1)/ Step;
// Кількість ітерацій на один потік
int H = (nFor + nTthreads - 1) / nTthreads;
for (i = 0; i < nTthreads; i++)
{
    Datas [i].Start = V1 + i * H;
    Datas [i].Finish = Datas [i].Start + H;
    Datas [i].Step = H;
    if (i != nTthreads - 1)
    hThread [i] =
    CreateThread (0, 0, ThreadFun, &Datas [i], 0, 0);
}    ThreadFun (&Datas [nTthreads - 1]);
WaitForMultipleObjects (nTthreads - 1, hThread, true, INFINITE);
```

НЕДОЛІКИ ОРГАНІЗАЦІЇ ПАРАЛЕЛЬНОГО ЦИКЛУ ЗА ДОПОМОГОЮ ФУНКЦІЙ WINDOWS

Тільки для циклів з відомою кількістю ітерацій.

Код для послідовного та паралельного виконання відрізняється.

Не враховується різниця в часі виконання ітерацій, якщо вона є.

Багато коду.

ВИСНОВКИ

1. Середовище розробки паралельних програм Open MP є кросплатформеною (windows, Unix).
2. Робота основана на використанні директив, тому один і той же текст програми можна використовувати для послідовного та паралельного виконання.
3. Треба уважно слідкувати за включенням режиму підтримки Open MP, в противному разі програма виконується послідовно
4. Для встановлення кількості паралельних потоків є багато способів, які дозволяють визначити кількість паралельних гілок для окремої ділянки програми, для всієї програми, або для усіх програм.
5. Паралельне виконання циклів за допомогою потоків Windows дуже складно.
6. Далі будуть розглянуті методи паралельного виконання циклу за допомогою технології Open MP.

ПИТАННЯ ДЛЯ САМОСТІЙНОГО ВИВЧЕННЯ

1. Функції Open MP для обчислення часу.
2. Директиви Open MP для UNIX подібних систем .

Учбовий посібник: “Паралельне програмування”

МАТЕРІАЛИ ДЛЯ ЕКСПРЕС-КОНТРОЛЮ

- Як включити режим підтримки Open MP?
- Як перевірити, чи включено режим підтримки Open MP?
- Як відключити паралельне виконання тільки однієї паралельної секції?
- Як визначити кількість потоків для однієї паралельної секції?
- Як визначити кількість паралельних потоків для усіх паралельних секцій програми?
- Як визначити кількість паралельних потоків для усіх програм?
- Які дані необхідно передати потоковій функції для паралельного виконання циклу?