Технологии параллельного программирования

Модуль 1. Технология OpenMP

Семинар 6

**Общие задания**

Скомпилировать и выполнить программный код на языке С++. Полученные результаты представить в отчете (скринами) преподавателю, сделать краткие выводы.

**Задание 1. Применение директивы for с различными опциями (варьировать значения опции type: static, dynamic, guided, auto, варьировать параметр опции type chunk, сравнить время)**

#include <iostream>

#include <stdio.h>

#include <omp.h>

#include <locale>

#include <windows.h>

using namespace std;

int main()

{

setlocale(LC\_ALL, "Russian");

int i;

double time = omp\_get\_wtime();

#pragma omp parallel private(i) num\_threads(4)

{

#pragma omp for schedule (static, 6)

//#pragma omp for schedule (dynamic, 6)

//#pragma omp for schedule (guided, 6)

//#pragma omp for schedule (auto)

for (i = 0; i<200; i++)

{

printf("Поток %d выполнила итерацию %d\n", omp\_get\_thread\_num(), i);

Sleep(1);

}

}

cout << "Time = " << (omp\_get\_wtime() - time) << endl;

}

**Задание 2. Решение задачи Фибоначчи 3-мя способами (последовательный, директива for, директива task)**

// последовательная

#include <iostream>

#include "omp.h"

using namespace std;

// Элемент списка

struct node {

int number; // Любое целое число

unsigned long int fib\_number; // Число Фибоначи

struct node\* next; // Указатель на следующий элемент списка

// node\* next; // Указатель на следующий элемент списка

};

// Рекурсивная функция вычисления n-го числа Фибоначи

int fibonacci(int n)

{

if (n < 2)

{

return (n);

}

else

{

return fibonacci(n - 1) + fibonacci(n - 2);

}

}

// Функция выполняющая независимое действие над элементом списка

void independent\_work(struct node\* list)

{

int n;

n = list->number;

list->fib\_number = fibonacci(n);

}

// Функция создания и инициализации списка

struct node\* init\_list(int n)

{

struct node\* list;

struct node\* head\_list = NULL;

struct node\* temp = NULL;

//node\* list;

//node\* head\_list = NULL;

//node\* temp = NULL;

head\_list = (struct node\*)malloc(sizeof(struct node));

//head\_list = new node();

list = head\_list;

list->number = 10;

list->fib\_number = 0;

for (int i = 1; i< n; i++) {

// temp = (struct node\*)malloc(sizeof(struct node));

temp = new node();

list->next = temp;

list = temp;

list->number = 10 + i;

list->fib\_number = 0;

}

list->next = NULL;

return head\_list;

}

int main(int argc, char \*argv[]) {

struct node \*list = NULL;

struct node \*temp = NULL;

struct node \*head\_list = NULL;

int count = 0;

const int n = 36; // Число элементов в списке

head\_list = init\_list(n); // Инициализация списка

double time = omp\_get\_wtime();

list = head\_list;

while (list != NULL)   // Основной цикл по выполнению действий над элементами списка

{

independent\_work(list);

list = list->next;

}

time = omp\_get\_wtime() - time;

list = head\_list;

while (list != NULL)  // Вывод элементов списка и освобождение памяти

{

printf("%d : %d\n", list->number, list->fib\_number);

temp = list->next;

free(list);

list = temp;

}

head\_list = NULL;

cout << "Time=" << time << endl;

}

// директива for

#include <iostream>

#include "omp.h"

using namespace std;

// Элемент списка

struct node {

int number;

unsigned long int fib\_number;

struct node\* next;

};

// Рекурсивная функция вычисления n-го числа Фибоначи

int fibonacci(int n)

{

if (n < 2)

{

return (n);

}

else

{

return fibonacci(n - 1) + fibonacci(n - 2);

}

}

// Функция выполняющая независимое действие над элементом списка

void independent\_work(struct node\* list)

{

int n;

n = list->number;

list->fib\_number = fibonacci(n);

}

// Функция создания и инициализации списка

struct node\* init\_list(int n)

{

struct node\* list;

struct node\* head\_list = NULL;

struct node\* temp = NULL;

head\_list = (struct node\*)malloc(sizeof(struct node));

list = head\_list;

list->number = 10;

list->fib\_number = 0;

for (int i = 0; i< n; i++) {

temp = (struct node\*)malloc(sizeof(struct node));

list->next = temp;

list = temp;

list->number = 10 + i;

list->fib\_number = 0;

}

list->next = NULL;

return head\_list;

}

int main(int argc, char \*argv[]) {

double start, end;

struct node \*list = NULL;

struct node \*temp = NULL;

struct node \*head\_list = NULL;

int count = 0;

const int n = 36;

head\_list = init\_list(n);

start = omp\_get\_wtime();

{

list = head\_list;

while (list != NULL) // Подсчет элементов списка

{

count++;

list = list->next;

}

list = head\_list;

node\*\* arrd = new node\*[count]; // Выделение памяти под массив, хранящий адреса элементов списка

for (int i = 0; i < count; i++) // Занесение в массив адресов элементов списка

{

arrd[i] = list;

list = list->next;

}

#pragma omp parallel num\_threads(4) // параллельная обработка элементов списка

{

#pragma omp for schedule(dynamic,6)

for (int i = 0; i < count; i++)

{

independent\_work(arrd[i]);

}

}

}

end = omp\_get\_wtime();

list = head\_list;

while (list != NULL)  // Вывод элементов списка и освобождение памяти

{

printf("%d : %d\n", list->number, list->fib\_number);

temp = list->next;

free(list);

list = temp;

}

free(list);

printf("Compute Time: %f seconds\n", end - start);

return 0;

}

// директива task

#include <iostream>

#include "omp.h"

using namespace std;

// Элемент списка

struct node {

int number;

unsigned long int fib\_number;

struct node\* next;

};

// Рекурсивная функция вычисления n-го числа Фибоначи

int fibonacci(int n)

{

if (n < 2)

{

return (n);

}

else

{

return fibonacci(n - 1) + fibonacci(n - 2);

}

}

// Функция выполняющая независимое действие над элементом списка

void independent\_work(struct node\* list)

{

int n;

n = list->number;

list->fib\_number = fibonacci(n);

}

// Функция создания и инициализации списка

struct node\* init\_list(int n)

{

struct node\* list;

struct node\* head\_list = NULL;

struct node\* temp = NULL;

head\_list = (struct node\*)malloc(sizeof(struct node));

list = head\_list;

list->number = 10;

list->fib\_number = 0;

for (int i = 0; i < n; i++) {

temp = (struct node\*)malloc(sizeof(struct node));

list->next = temp;

list = temp;

list->number = 10 + i;

list->fib\_number = 0;

}

list->next = NULL;

return head\_list;

}

int main(int argc, char\* argv[]) {

double start, end;

struct node\* list = NULL;

struct node\* temp = NULL;

struct node\* head\_list = NULL;

int count = 0;

const int n = 36;

head\_list = init\_list(n);

start = omp\_get\_wtime();

#pragma omp parallel  // параллельная обработка элементов списка  с использованием task

{

#pragma omp single nowait

{

list = head\_list;

while (list) {

#pragma omp task firstprivate(list)

{

independent\_work(list);

}

list = list->next;

}

}

}

end = omp\_get\_wtime();

list = head\_list;

while (list != NULL)  // Вывод элементов списка и освобождение памяти

{

printf("%d : %d\n", list->number, list->fib\_number);

temp = list->next;

free(list);

list = temp;

}

free(list);

printf("Compute Time: %f seconds\n", end - start);

return 0;

}

**Задания для самостоятельного выполнения**

1. Написать пример реализации директивы For с опцией reduction, в котором определенным образом накапливаются значения из разных итераций цикла. Проиллюстрировать работу.
2. Написать пример реализации директивы Sections и директивы Section (совместно) для 4-х задач, в которых все задачи могут работать параллельно, но 2 и 3 задачи – только вместе, друг за другом.