Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Кафедра автоматизированных систем управления (АСУ)

ОБРАБОТКА И ВЫПОЛНЕНИЕ ПРОГРАММ В СРЕДЕ OpenMP

Лабораторная работа №4

по дисциплине

«Параллельное программирование»

Студент гр. 430-2

_____А.А. Лузинсан

«____»_____2023 г.

Руководитель

Ассистент кафедры АСУ

_____ П.Д. Тихонов

«____»_____2023 г.

Томск 2023

Оглавление

Введение

Цель работы: освоить применение основных директив, функций и переменных окружения OpenMP на примере параллельной программы численного интегрирования.

Индивидуальное задание по варианту №6-n:

$$\int_{-0.2}^{1.0} \frac{dx}{1+(c*x)^2} = \frac{1}{c} arctg(c*x); \quad c = 0.9$$

1 Ход работы

1. Последнюю программу лабораторной работы вычисления интеграла: integi.c или integn.c. скопировать в новый файл. Проанализировать функции MPI для включении аналогичных директив и функций OpenMP в программу. Убрать из программы функции MPI и сделать последовательную программу. Отметить места функций MPI для последующего включения аналогичных конструкций OpenMP. Выполнить последовательную программу. Проверить совпадение результатов с параллельной программой.

Первым делом был скопирован листинг последней версии программы по первой лабораторной работе. Далее, программа была модифицирована так, что из неё были убраны все участки включения функций MPI. Помимо этого программа была переделана на работу только в последовательном режиме. Результаты совпадали с результатами выполнения программы первой лабораторной работы.

2. Спланировать параллельные регионы программы. Определить набор общих и локальных переменных. Назначить переменную для редукции.

В программе были определены публичные переменные: intervals, xl, xh, c, step. Локальной переменной выступает переменная цикла i. В качестве переменной для редукции была назначена переменная integral, в которую в цикле считается частная сумма и в последующем суммируются все частные суммы потоков.

3. Добавить в программу директивы parallel и for с необходимыми параметрами и типами переменных. Распределение итераций цикла не планировать. Полученные в отдельных потоках частные суммы собрать переменной редукции в общую сумму для значения интеграла.

Перед определением цикла в программу была добавлена директива parallel for с параметрами shared и reduction. Директива parallel предназначена для определения параллельного фрагмента в коде программы. Директива for

предназначена для распараллеливания циклов. Параметр shared определяет общие переменные для всех имеющихся потоков, тогда как параметр reduction задаёт операцию свёртывания локальных переменных через заданный оператор.

4. Обработать и выполнить параллельную программу OpenMP с числом процессов, установленных в системе.

Число потоков для выполнения параллельных частей программы в операционной системе определяется переменной окружения OMP_NUM_THREADS. Приоритет способа задания числа потоков выстроен следующим образом: параметр num_threads в директиве parallel, функция omp_set_num_threads(), переменная окружения OMP_NUM_THREADS. Если ничего из перечисленного не установлено, то по-умолчанию берётся максимально доступное кол-во потоков, определить которое можно с помощью функции omp_get_max_threads().

5. Включить в программу функции OpenMP и вывести значения переменных интерфейса: максимально возможное число потоков — omp_get_max_threads(), установка числа потоков в параллельной области — omp_set_num_threads(n) , определение числа потоков в параллельной области — omp_get_num_threads(), номер каждого потока параллельного региона — omp_get_thread_num(), время работы программы — omp_get_wtime().

В программу были включены функции OpenMP omp_get_max_threads(); omp_set_num_threads(n); omp_get_num_threads() в директиве master параллельной области; omp_get_thread_num() и omp_get_wtime().

6. Включить в директиву FOR параметр распределения итераций (schedule). Размер блока итераций (chunk) выбрать таким, чтобы в каждом потоке выполнялось несколько частей итераций цикла. Сравнить время выполнения программы в статическом (static) и динамическом (dynamic) режимах.

Далее в список параметров директивы parallel for был включен

параметр shedule. В качестве опции планирования распределения итераций между потоками было рассмотрено 2 алгоритма: static и dynamic — с различным значением количества блоков. Показатели затраты времени для каждого режима представлены в таблице 1.1.

• schedule(static) — статическое планирование, где итерации цикла делятся приблизительно поровну между потоками. В результате выполнения цикла в таком режиме время выполнения программы заняло 5.346154e-01 сек.

• schedule(static, 1) — блочно-циклическое распределение итераций, где каждый поток получает указанное число итераций в начале цикла. Затем процедура распределения продолжается по такой же схеме. Данная идея распределения была в первой лабораторной работе. Планирование выполняется один раз, при этом каждый поток узнаёт итерации, которые должен выполнить. Время выполнения программы, содержащий распараллеленный в таком режиме цикл, прошло за 5.313556e-01 сек.

• shedule(static, 100) — описание представлено выше. В результате выполнения цикла с данным значением CHUNK время выполнения программы заняло 5.435105e-01 сек.

• schedule(dynamic) — динамическое планирование со значением по-умолчанию 1, где каждый поток получает заданное число итераций, выполняет их и запрашивает новую порцию итераций. В отличие от статического планирования, планирование выполняется многократно и конкретное распределение итераций между потоками зависит от темпов работы потоков и трудоёмкости итераций. Время выполнения: 1.657966e+00 сек.

• schedule(dynamic, CHUNK). Время выполнения: при CHUNK=100 time=5.135971e-01; при CHUNK=1000 time=5.069887e-01.

7. Доработать программу назначением числа нитей параллельной части программы параметром командной строки при запуске программы.

В программу была добавлена возможность указания количества потоков из командной строки. Таким образом, при указании оного, вызывается функция opm_set_num_threads(), куда далее передаётся целочисленное значение из списка параметров запроса.

8. Выполнить программу для разного числа нитей и записи результатов в текстовые файлы. Попробуйте менять значения параметра распределения итераций в цикле.

Наконец, программа была запущена на различном количестве потоков в разных режимах. Итоговая сводка по временным затратам представлена в таблице 1.1.

Таблица 1.1 — Времязатраты для каждого режима распределения итераций на различном количестве потоков параллельных регионов

Режим / Потоки	static, auto	static, 1	static, 100	static, 1000	dynamic, 1	dynamic, 100	dynamic, 1000
1 thread	2.36E+00	2.38E+00	2.37E+00	2.44E+00	5.17E+00	2.49E+00	2.38E+00
2 threads	1.21E+00	1.23E+00	1.22E+00	1.21E+00	2.97E+00	1.24E+00	1.20E+00
4 threads	6.87E-01	6.65E-01	6.83E-01	6.60E-01	2.76E+00	6.73E-01	6.78E-01
8 threads	5.75E-01	5.23E-01	5.18E-01	5.22E-01	1.64E+00	5.17E-01	5.15E-01

Как видно из таблицы 1.1, оптимальным режимом распределения задач является schedule(dynamic, 1000).

2 РЕЗУЛЬТАТЫ РАБОТЫ ПРОГРАММЫ

В результате запуска программы с указанием различного количества потоков, листинг которой представлен в приложении А.1, было выявлено, что оптимальным режимом для заданного количества интервалов (10^8) является распределение schedule(dynamic, 1000). Для данного режима результаты на различном числе процессов представлены на рисунке 2.1.

```
----------------------DYNAMIC 1000--------------------------

Number of intervals: 100000000
Maximum number of threads: 8
Evaluation on 1 threads
Step: 1.2e-08

DYNAMIC chunk:1000
Integral is approximately: 1.012120e+00
Error: 5.562217e-14
Time of calculation: 2.384546e+00

------------------------------------------------------------
Number of intervals: 100000000
Maximum number of threads: 8
Evaluation on 2 threads
Step: 1.2e-08

DYNAMIC chunk:1000
Integral is approximately: 1.012120e+00
Error: 2.009504e-14
Time of calculation: 1.204010e+00

------------------------------------------------------------
Number of intervals: 100000000
Maximum number of threads: 8
Evaluation on 4 threads
Step: 1.2e-08

DYNAMIC chunk:1000
Integral is approximately: 1.012120e+00
Error: -2.248202e-13
Time of calculation: 6.778650e-01

------------------------------------------------------------
Number of intervals: 100000000
Maximum number of threads: 8
Evaluation on 8 threads
Step: 1.2e-08

DYNAMIC chunk:1000
Integral is approximately: 1.012120e+00
Error: 3.363976e-14
Time of calculation: 5.149840e-01

```

Рисунок 2.1. - Результаты выполнения программы с указанием различного количества потоков для параллельных регионов

Заключение

В результате выполнения лабораторной работы я освоила применение основных директив, функций и переменных окружения OpenMP на примере параллельной программы численного интегрирования.

Приложение А
(обязательное)
Листинг программы

Листинг А.1 — Листинг программы с параллельными регионами для задачи численного интегрирования

```
#include <omp.h>
#include <iostream>
#include <math.h>
#include <fstream>
#define CHUNK 1000
static double f(double a, double c);
static double fi(double a, double c);

void lab4(int argc, char *argv[])
{
   std::ofstream fout("/home/luzinsan/TUSUR_learn/4 курс/7_semester/ПП/labs/proj_all/bin/4/output.txt", std::ios::app);
   fout << "\n\n------------------------------------------------------------";
   int intervals = 100000000;
   fout << "\nNumber of intervals: " << intervals;
   double xl=-0.2, xh=1.0, c=0.9;
   int max_threads = omp_get_max_threads();
   fout << "\nMaximum number of threads: " << max_threads;
   if (argv[1])
      omp_set_num_threads(std::stoi(argv[1]));
   #pragma omp parallel
   #pragma omp master
      fout << "\nEvaluation on " << omp_get_num_threads() << " threads";
   double integral = 0;
   double step = (xh - xl) / static_cast<double>(intervals);
   fout << "\nStep: " << step << '\n';
   double startwtime = omp_get_wtime();
   fout << "\nSTATIC chunk:" << CHUNK;
   // fout << "\nSTATIC auto";
   // fout << "\nDYNAMIC chunk:" << CHUNK;
   // fout << "\nDYNAMIC auto";

   #pragma omp parallel for shared(step, xl, c) reduction (+: integral)\
            schedule(static, CHUNK)
      for (int i = 1; i <= intervals; i++)
      {
         double x = xl + step * ((double)i - 0.5);
```

```
            integral += f(x, c) * step;
        }
    fout << std::scientific << "\nIntegral is approximately: " << integral;
    fout << std::scientific << "\nError: " << integral - fi(xh, c) + fi(xl, c);
    fout << std::scientific << "\nTime of calculation: "
         << omp_get_wtime() - startwtime;
}

int main(int argc, char *argv[])
{
    lab4(argc, argv);
    return 0;
}

static double f(double a, double c)
{ return 1 / (1 + pow(c * a, 2)); }

static double fi(double a, double c)
{ return (1 / c) * atan(c * a); }
```